Aerodynamic Simulations of a Helicopter Rotor Using an Iterative Greedy Multiscale Radial Basis Function Mesh Deformation Method

A thesis submitted to fulfil the requirements of the degree of Doctor of Philosophy

Alexander Luke Coyle

Supervisor

Prof. Ben Thornber

2026

The University of Sydney

School of Aerospace, Mechanical and Mechatronic Engineering

Statement of Originality

I certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any other degree or purpose.

I certify that the intellectual content of this thesis is the product of my own work, and that all assistance received in preparing this thesis and all sources have been acknowledged.

Statement on the Use of Generative Artificial Intelligence

During the preparation of the thesis the author queried ChatGPT as a reference during the coding of a small number of subroutines for basic functions, such as file input and output, graphing and geometric data processing. It was also used to assist in the selection of a very small number of words from descriptions of the desired meanings.

Authorship Attribution Statement

This thesis contains material previous presented in the conference paper *Actuator Surface Model and Blade Resolved Simulations of an Attack Reconnaissance Class Rotor in Hover and Forward Flight* by A Coyle and B Thornber at the 49$^{\text{th}}$ European Rotorcraft Forum 2023.

This material includes Figures 50-54, as well as previous versions of Figures 40, 44 and 81. Tables 13 and 12, and a variant of Table 20 were also presented. The discussions in the conference paper include parts of the writing in Sections 2 and 6.2, and a minority of the discussions in Sections 5.1, 5.2, 6.3 to 6.6 and 7.6.

Across this work, I performed the review of literature, applied the ASM to the simulation case with assistance from Dr Daniel Linton, performed the blade-resolved simulations, analysis of the results and writing of the conference paper. The process was done with direction from Dr Ronny Widjaja, and generally with assistance from and under the supervision of Prof Ben Thornber.

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Acknowledgements

I would firstly like to thank my supervisor Prof. Ben Thornber. Ben, I have valued your endless patience, guidance, support and encouragement so highly. You have offered your time and assistance so generously in Sydney, online and in Dublin, whether my issues be technical or on the personal side of university and research life. I have been incredibly fortunate to have had you supervise my PhD.

I have also had the pleasure of working with a number of bright and helpful individuals who have generously gifted their time, and provided assistance and direction at various stages of my research. I would like to extend a particular thanks to Dr Ronny Widjaja, Dr Adam Murray, Dr Daniel Linton and Dr Michael Groom. I'd also like to thank everyone with whom I've shared the office. Your companionship has been a great support and you provided a fun relief from work.

I am also extremely thankful for all of those closest to me in my personal life. I can't express how grateful I am for your endless love, support, patience, understanding and encouragement that you have shown throughout the highs and lows of my studies. This endeavour was only possible with you all beside me. I love you very much.

This research was supported by an Australian Government Research Training Program (RTP) Scholarship and additionally The Defence Science and Technology Group Rotor Noise Scholarship.

Abstract

Within the area of aerodynamic simulations, there are innumerable scenarios which include the presence of moving or deforming bodies. There are a plethora of methods that have been developed to allow the simulation of such phenomena, with three large classes of methods being Immersed Boundary Methods, chimera grids, and Arbitrary Lagrangian-Eulerian methods. Within the Arbitrary Lagrangian-Eulerian methods, there has been growing interest in Radial Basis Function mesh deformation techniques. A Radial Basis Function (RBF) is a function with an output that is only dependant on the distance between two points. RBF mesh deformation designates base nodes, a set of nodes with known positions before and after the deformation. The displacement of these base nodes and the chosen RBFs are then used to map a displacement field to the rest of the domain. This group of methods are favourable due to them not requiring mesh connectivity data, producing high quality meshes post-deformation, and performing many of the necessary calculations as preprocessing.

This thesis presents a new RBF mesh motion algorithm which combines and expands upon a number of techniques that have been proven successful. The mechanics of the mesh deformation are based on the multiscale RBF method, with base node selection accomplished through a greedy algorithm. Iterative methods were then applied to this greedy multiscale method in order to minimise the preprocessing time. This combination of RBF approaches provides excellent deformed grid quality at a preprocessing cost which is inconsequential compared to the computational cost of the simulation as a whole. Furthermore, as a result of the greedy algorithm, this method involves little user input, with the process of choosing base nodes no longer necessary.

RBF methods are examined in application to aerodynamic simulations of helicopter rotors. First, the multiscale method is used for collective pitch adjustment in hover, with a very low change in grid quality seen over a range of -4° to 4° changes in collective angle. The aerodynamic results from the simulations run on these meshes are confirmed to be similarly accurate to a number of high quality blade-resolved flow solvers and the Actuator Surface Model, a hybrid Computational Fluid Dynamics method. Following this, the iterative, greedy, multiscale algorithm is investigated. Analysis of preprocessing times confirm the mathematical analysis of the iterative algorithms, with significant time saving over the noniterative algorithms. The grid quality is compared between the multiscale method and the new algorithm, with a noticeable increase in grid quality displayed across a number of measures.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

A_{ijk}	Area of triangle ijk
a	Timing fit exponent
$\mathbf{B}$	Base set coordinate matrix
$\mathbf{b}$	Base node coordinate vector
b	Timing fit multiplicative coefficient
c	Chord length
c_{loc}	Local chord length
C_D	Drag coefficient
C_i^{ijk}	Torsional spring stiffness at node i of triangle ijk
C_L	Lift Coefficient
C_Q	Torque coefficient
C_q	Local torque coefficient
C_T	Thrust coefficient
C_t	Local thrust coefficient
$\mathscr{D}$	Spalart-Allmaras destruction term
d	wall distance
$\mathbf{E}$	Greedy method error vector
$\mathbf{F}$	Total RBF influence vector
F	Total RBF influence
$\mathbf{f}$	Source vector
FoM	Figure of Merit
$\mathbf{H}$	Base set displacement to volume node displacement matrix
$\mathbf{I}_i$	$i \times i$ identity matrix
i_E	Index of maximum error value in $\mathbf{E}$
i_M	Index of maximum separation distance in separation distance vector
J	Cell Jacobian
$\mathscr{J}$	Jacobian of curvilinear coordinate transformation
k	Turbulent kinetic energy
K_η	Mach number correction factor
k_{ij}	Spring stiffness between nodes i and j
$\mathbf{k}$	Diffusion coefficient function
l_{ij}	Length of edge ij
$\mathbf{L}$	Refinement node to refinement node influence submatrix of multiscale influence matrix
M	Mach number
M_n	Normal mach number
N_b	Number of base nodes
N_r	Number of refinement nodes
N_{red}	Number of nodes in reduced base node set
N_s	Number of surface nodes
N_v	Number of nodes in the volume mesh
$\mathbf{o}_r$	Refinement node order vector
P	Pressure

$\mathscr{P}$	Spalart-Allmaras production term
$\mathbf{p}$	New refinement node position vector
p^i	Polynomial function for total RBF influence function
Pr	Prandtl number
$\mathrm{Pr_t}$	Turbulent Prandtl number
Q	Global orthogonality
q	Local orthogonality
q_i	Local orthogonality in i plane
q_j	Local orthogonality in j plane
q_k	Local orthogonality in k plane
$\mathbf{R}$	Refinement set coordinates matrix
R	Rotor radius
$\mathscr{R}$	Spalart-Allmaras trip term
Re	Reynolds number
Re_y	Yawed Reynolds number
$\mathbf{r}$	Refinement node coordinate vector
$\mathbf{s}$	Surface node coordinate vector
T	Rotor thrust
$\mathscr{T}$	Temperature
t	Time
$\mathbf{V}$	Volume node set coordinate matrix
$\mathbf{V}_2$	Category 2 volume node set coordinate matrix
$\mathbf{V}_3$	Category 3 volume node set coordinate matrix
$\mathbf{v}$	Volume node coordinate matrix
$\mathbf{v}_a$	Adjacent volume node coordinate matrix
$\mathbf{w}$	Mesh velocity
$\mathbf{x}$	Coordinate vector
$\mathbf{x}_0$	Radial basis function centre coordinate vector
$\mathbf{x}_{\mathrm{red}}$	Reduced base node set coordinate vector
x	X coordinate
Y	Y coordinate
z	Z coordinate
$\boldsymbol{\alpha}$	Base node Radial Basis Function coefficient vector
$\boldsymbol{\alpha}_\Delta$	Base node displacement Radial Basis Function coefficient vector
α	Base node Radial Basis Function coefficient
α_s	Rotor shaft angle
$\boldsymbol{\beta}$	Refinement node Radial Basis Function coefficient vector
β	Refinement node Radial Basis Function coefficient
β_0	Coning angle
Γ	Vector of primitive variables
γ	Ratio of specific heats
θ	Blade pitch angle
θ_0	Collective pitch angle
θ_{1c}	Lateral cyclic pitch angle
θ_{1s}	Longitudinal cyclic pitch angle
θ_i^{ijk}	Angle at vertex i in triangle ijk

κ	Von Kármán constant
$\varkappa$	Matrix condition
Λ	Flow yaw angle
μ	Dynamic viscosity
μ_0	Reference dynamic viscosity
μ_t	Turbulent dynamic viscosity
ν	Kinematic viscosity
$\tilde{\nu}$	Spalart-Allmaras working variable
$\boldsymbol{\Xi}$	Vector of separation distances
Ξ	Separation distance
ξ	Curvilinear coordinate variable
ρ	Density
ϱ	Ratio of variation
σ	Rotor solidity
τ	Viscous stress
τ_w	Wall shear stress
μ_f	Advance ratio
$\boldsymbol{\Phi}$	Base set to base set influence matrix
$\boldsymbol{\Phi}_b$	Base set to base set influence submatrix of multiscale influence matrix
$\boldsymbol{\Phi}_r$	Base set to refinement set influence submatrix of multiscale influence matrix
ϕ	Radial Basis Function
φ	Flux limiter function
χ	Variable for Spalart-Allmaras turbulence model: $\tilde{\nu}/\nu$
$\boldsymbol{\Psi}$	Base set to volume node influence matrix
$\boldsymbol{\Psi}_b$	Base set to volume node influence matrix, multiscale system
$\boldsymbol{\Psi}_{b,v2}$	Base set to category 2 volume node influence matrix
$\boldsymbol{\Psi}_{b,v3}$	Base set to category 3 volume node influence matrix
$\boldsymbol{\Psi}_r$	Refinement set to volume node influence matrix
$\boldsymbol{\Psi}_{r,v3}$	Refinement set to category 3 volume node influence matrix
ψ_r	Rotor blade azimuth angle
Ω	Rotor rotation speed
ω	Specific rate of turbulent kinetic energy dissipation
$\eth$	New curvilinear transform time variable

Acronyms

ALE	Arbitrary Lagrangian-Eulerian
AMR	Adaptive Mesh Refinement
ARC	Attack Reconnaissance Class
ASM	Actuator Surface Model
BERP	British Experimental Rotor Programme
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
CSR	Compressed Sparse Row
DES	Detached Eddy Simulation
DG-RBF	Delaunay Graph Mapping and Radial Basis Function
DLB	Dynamic Load Balancing
DNS	Direct Numerical Simulation
HMB3	Helicopter Multi-Block
IB	Immersed Boundary
IBM	Immersed Boundary Method
LES	Large Eddy Simulation
OpenFOAM	Open source Field Operation and Manipulation
PDE	Partial Differential Equation
PISO	Pressure-Implicit with Splitting of Operators
RANS	Reynolds Averaged Navier Stokes
RBF	Radial Basis Function
RCAS	Rotorcraft Comprehensive Analysis System
SBR	Solid Body Rotation
TFI	Transfinite Interpolation

1 Introduction

1.1 Background

The field of engineering has always been striving to produce the most accurate and efficient calculation and simulation tools possible in an effort to streamline and improve design. The area of aerodynamic calculations has seen rapid developments with the introduction and improvement of computing. This has enabled increasingly detailed simulations, improving the fidelity of the fluid dynamics predictions, as well as adding coupled areas of physics to these calculations. One crucial aspect of aerodynamic simulations is the incorporation of moving and deforming bodies. It is common for aerodynamic simulations to involve bodies that move relative to each other, for example helicopters, wind turbines and motor vehicles. Additionally, aeroelastic effects are inescapable in a number of scenarios, such as in wing design. Aeroelastic phenomena can be dramatic, as can be seen in Figure 1, which shows the significant deformation of the wings of a DG-300 during a flutter test. Such scenarios with multiple moving bodies or deforming objects cannot be simulated by varying the flow conditions, as may be possible with, for instance, an oscillating aerofoil. Therefore, additional techniques must be introduced to Computational Fluid Dynamics codes. Techniques that can enable such simulations include Immersed Boundary Methods, chimera grids and Arbitrary Lagrangian-Eulerian methods. Immersed boundary conditions have seen a number of recent developments, but are still complex and require significant computing resources. Chimera grids are very effective for moving objects, but do not in themselves allow for the deformation of bodies being simulated. Arbitrary Lagrangian-Eulerian methods simulate flow on a single boundary conforming grid, which must correspondingly be deformed with the object motion. A vast number of methods have been developed for this purpose.

One of these Arbitrary Lagrangian-Eulerian methods is Radial Basis Function (RBF) mesh deformation. This method employs radial basis functions, functions whose output is a function of only the distance between two points, to define a deformation field determined by the movement of some chosen nodes. A basic example of RBF deformation is presented in Figure 2. The movement of the controlling node creates a displacement field within the red dotted circle, the strength of which is proportional to the distance from the controlling node's original location. The mesh nodes are moved in a manner corresponding to this field strength, with the mesh nodes closest to the controlling nodes moving the most.

RBF methods are appealing for a number of reasons. Firstly, RBF methods require no mesh connectivity data, being able to operate solely on node coordinates. Secondly, RBF methods perform much of the necessary calculations in preprocessing, reducing the per-timestep cost of the flow solver. Thirdly, these methods have been shown generally produce high quality mesh deformations across a variety of applications. One important aspect of this is that the mesh quality is maintained very well close to the surface of the moving or deforming object. As the simulation accuracy is often critical in this near-object region, in particular in boundary layers, this makes it more likely that the accuracy of the aerodynamic predictions are better

Figure 1: An image from a flutter test of the DG-300 [1]

maintained.

However, there are still challenges in the application of RBF mesh deformation methods. RBF methods are computationally expensive, to the point where the preprocessing can become impractical with the number of nodes used in modern meshes. This has lead to development of methods that use subsets of the surface nodes to drive the mesh deformation, effectively reducing the system size. However, the selection of the deformation-controlling nodes, known as base nodes, now becomes an important challenge. These base nodes should be chosen so that the mesh near the objects' surfaces is adequately "influenced" by the base nodes' motion, whilst also producing a well conditioned system.

This thesis presents a novel method which addresses the issue of base node selection whilst maintaining high deformation quality and reasonable calculation times. This method combines the multiscale RBF method presented by Kedward et al. [2] with a greedy point selection algorithm. This means that the only necessary inputs in base node selection are the number of nodes, and the deformation and movement that the base node selection should be optimised for. The preprocessing of this method is kept within reasonable costs through the use of iterative sub-algorithms within the greedy point selection. These reformulate and greatly expand on the methods presented by Zhao [3]. In total, this new method provides a near-optimal base node selection for a multiscale RBF implementation with a small computational cost in relation to the total simulation. This simplifies the preprocessing stage of the simulations while also maximising the ratio of mesh quality to the number of base nodes.

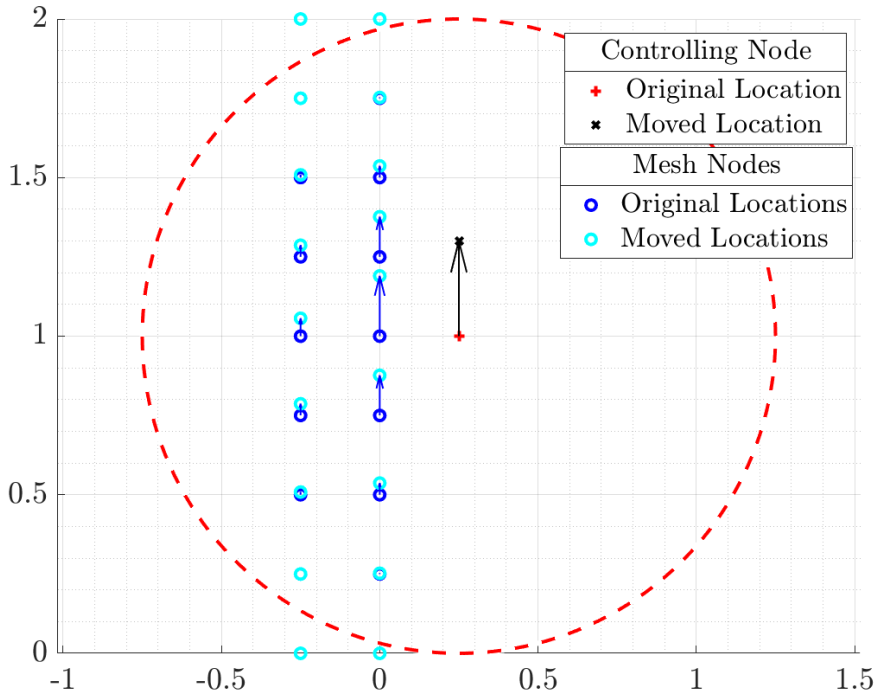


Figure 2: A simple example of RBF mesh deformation, the blue mesh nodes are moved according the controlling node, with the movement being a function of the distance to the controlling node in the undeformed mesh

Not enough of problem statement?? More emphasis on challenges of current RBF methods?

1.2 Thesis Outline

This thesis is comprised of six chapters, excluding introductory and concluding material. These progress from an overview of mesh motion methods and background of the simulations performed, to a presentation of the methodologies used in these simulations and an analysis of the results. Chapter 2 gives a background to the rotor aerodynamics of the Boeing AH-64 Apache as the test case for the solvers and methodologies found in this thesis. This includes details of the blade, an introduction to helicopter rotor performance parameters, and a summary of aerodynamic studies on this rotor.

Chapter 3 gives an overview of methods for simulating flows around moving objects, with a focus on Radial Basis Function (RBF) techniques. This begins with defining a desired set of characteristics for these methods. Chimera and Immersed Boundary methods are then discussed, before examining a number of popular Arbitrary Lagrangian-Eulerian methods. RBF techniques are then explored in detail,

beginning with the fundamental mathematics underpinning these methods, progressing to recent developments in RBF methods, and then compared to other Arbitrary Lagrangian-Eulerian methods. These techniques are all evaluated with regards to the characteristics set out at the beginning of the chapter.

Chapter 4 details the RBF methodologies specifically used in this thesis, with a focus on the actual implementation. In this section, a novel iterative, greedy, multiscale RBF algorithm is presented as an advancement of RBF methods. This method achieves very high quality mesh deformation with a near-optimal quality to cost ratio, whilst also maintaining reasonable computational requirements. The use of variable base node support radii is also proposed and justified with a particular focus on application to rotor simulations.

Chapter 5 outlines the methods employed by the two CFD solvers that are used to perform the simulations in this thesis. The first of these solvers is the Actuator Surface Model (ASM), which is a hybrid CFD method that represents the blade presence though momentum sources rather than an explicit representation in the mesh. The second solver used in this thesis is Flamenco, an in house CFD solver of The University of Sydney. Flamenco is a structured, curvilinear, Godunov-type finite volume code which solves the compressible Navier-Stokes equations with high order schemes.

Chapters 6 and 7 detail the results of the rotor aerodynamics simulations. Chapter 6 covers the simulations of rotors in hover, examining the grid deformation quality, the integrated and sectional loads, and an analysis of flowfield features. Chapter 7 presents the forward flight simulation results with a focus on the iterative greedy multiscale RBF method. This analysis includes a comparison of preprocessing times between iterative and noniterative methods, a comparison of grid quality metrics to the multiscale method, and an analysis of the aerodynamic simulation results.

Finally, Chapter 8 concludes with an overview of the methods and results presented in the previous chapters. It also presents suggestions for future research that could expand on the work presented in this thesis.

2 Boeing AH-64 Apache Rotor Aerodynamics

The novel greedy multiscale RBF point selection algorithm is demonstrated in application to simulations of helicopter rotor aerodynamics. Specifically, the rotor aerodynamics of an Attack Reconnaissance-Class Helicopter in both hover and forward flight is compared to experimental results, as well as a number of simulation tools. This chapter provides an overview of the Attack Reconnaissance-Class Helicopter, its blade geometry, helicopter rotor performance metrics, and previous literature on the simulation of the rotor.

2.1 Attack Reconnaissance-Class Helicopter

The Attack Reconnaissance-Class helicopter is based on the Boeing AH-64 Apache helicopter. This is an attack helicopter that is currently in use by 16 countries [4]. The Apache project began with the first flight of the YAH-64 prototype, which itself was heavily based on the Hughes Helicopters Model 77. The delivery of the first helicopter was to the United States Army in 1986. Since then there have been a number of updates to the machine, with the most recent AH-64H variant beginning full scale production in 2012.

The AH-64 Apache was chosen as the base for this project mainly due to the presence of favourable data for the validation of the solvers used in this thesis. There is a high quality dataset for the performance and aerodynamics of the AH-64 rotor in both hover and in forward flight, as outlined in Section 2.3. This includes experimental data, as well as simulation data produced by a range of tools from lifting line codes to high-order Navier-Stokes solvers. With such a range of data, the validation of the results produced using the methods presented in this thesis can be conducted thoroughly.

The ARC blade is based on the Apache blade geometry and has a radius $R = 288in\,(7.315m)$ and a chord $c = 21in\,(0.5334m)$. Figure 3 below shows the geometric profile with measurements in inches.

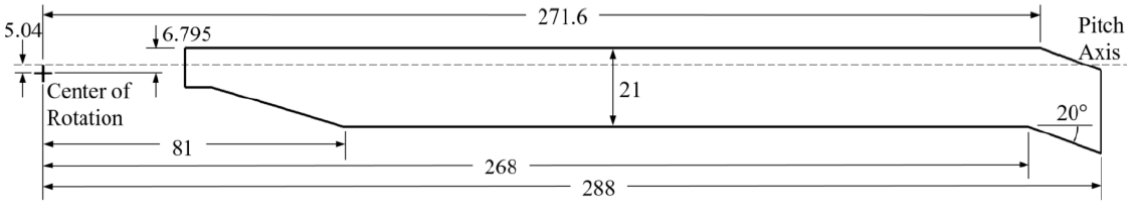


Figure 3: Definition of the ARC class blade (measurements in inches) [5]

The blade has a -9° linear twist. The section from blade root to $0.94R$, where the leading edge sweep begins, uses the Hughes Helicopters HH-02 aerofoil. The tip aerofoil profile is a NACA64A006, and the swept region from $0.94R$ outwards correspondingly transitions between the two profiles. The HH-02 aerofoil is plotted

with a representative chord of 1 in Figure 4, with the coordinates taken from Hu [6].

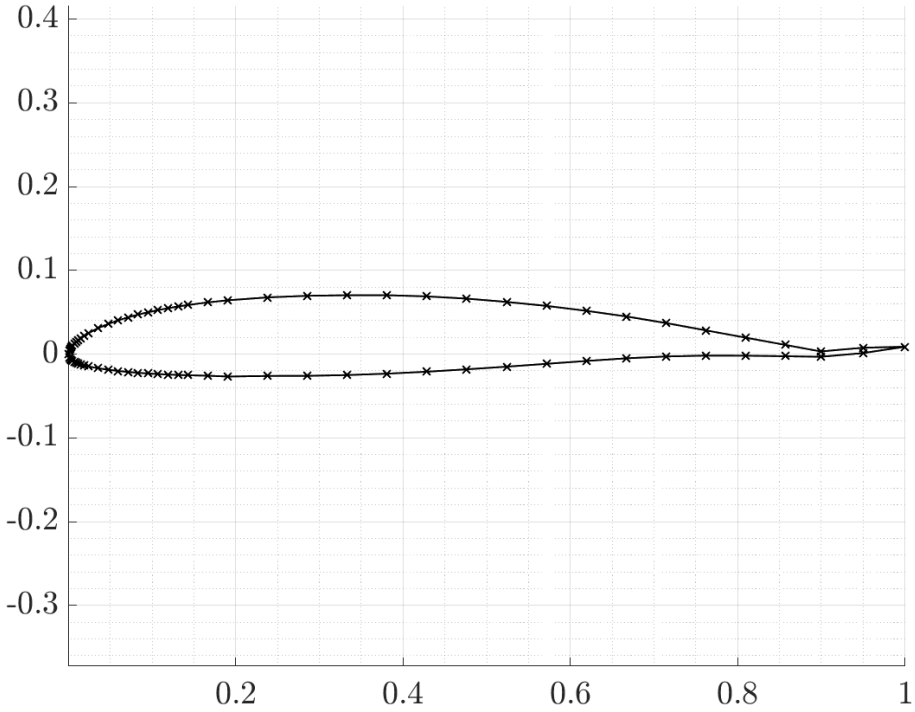


Figure 4: HH-02 Aerofoil Profile

The NACA64A006 aerofoil is similarly plotted with coordinates from Stivers in Figure 5 [7].

2.2 Helicopter Rotor Performance and Parameters

The following definitions are valuable in quantitative analysis of helicopter rotor performance and simulations. While some of the definitions are conceptually familiar to those in the aeronautics and aerodynamics fields, the mathematical definitions in the context of helicopter rotors can be slightly different to the more well known definitions for non-rotating bodies.

The thrust coefficient is a nondimensionalised measure of the total thrust generated by the rotor:

$$C_T = \frac{T}{\rho_\infty (\Omega R)^2 \pi R^2} \tag{1}$$

where T is the thrust generated by the rotor, ρ_∞ is the freestream density, Ω is the rotation speed of the rotor and R is the rotor radius.

The torque coefficient is a similarly defined, nondimensional measure of the torque experienced by the rotor:

$$C_Q = \frac{Q}{\rho_\infty (\Omega R)^2 \pi R^3} \tag{2}$$

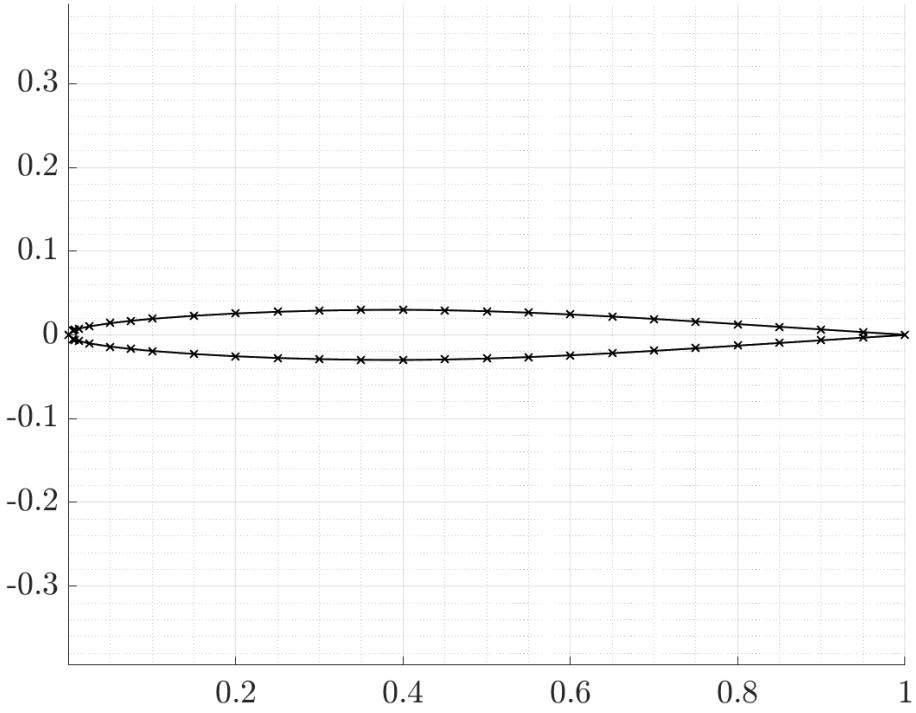


Figure 5: NACA64A006 Aerofoil Profile

where Q is the total torque on the rotor.

The figure of merit (FoM) is essentially a measure of the efficiency of the rotor. It is the ratio of the power consumed in the production of lift to the total power required by the rotor [8, 9], effectively comparing the ideal power required for hover to the actual power consumed [10]. The FoM can be defined using C_T and C_Q:

$$\text{FoM} = \frac{C_T^{\frac{3}{2}}}{\sqrt{2}C_Q} \tag{3}$$

There are also sectional thrust and torque coefficients C_t and C_q. These describe the thrust and torque generated at a specific radial section of a rotor blade. These are defined as:

$$C_t = \frac{dT}{dr}\frac{2}{\rho_\infty c_{loc}(\Omega r)^2} \tag{4}$$

$$C_q = \frac{dQ}{dr}\frac{2}{\rho_\infty c_{loc} R(\Omega r)^2} \tag{5}$$

where c_{loc} is the local chord and r is the radial station of the slice. The values $\frac{dT}{dr}$ and $\frac{dQ}{dr}$ can be considered as the sectional trust and torque generated by a section of the blade with an infinitesimally small radial width. Practically, this is typically evaluated as the thrust or torque generated by a section of a blade divided by the width of this section.

The solidity of the rotor is a measure of the area of the rotor disk taken up by the blades compared to the total area of the rotor disk. This is defined mathematically as:

$$\sigma = \frac{Nc}{\pi R} \tag{6}$$

where N is the number of blades.

2.3 Simulations of the Attack Reconnaissance Class Helicopter

There are a number of simulations and experiments which capture the airflow around and the performance of the Boeing AH-64 Apache rotor, therefore providing a well established validation case to confirm the effectiveness of the Flamenco flow solver with the new iterative greedy multiscale RBF mesh motion technique. This section provides an overview of this research, highlighting the simulation cases, CFD solvers and mesh details.

Narducci and Tadghighi of Boeing used the Helios CFD code to simulate the flow around the AH-64E Apache in hover and forward flight [11]. CREATE$^{\text{TM}}$-AV Helios is a multidisciplinary code for rotary wing aircraft simulations developed by the Department of Defense High Performance Computing and Modernization Office of the USA. This code consists of a number of modules: NSU3D, an unstructured solver suited for challenging-to-mesh geometries such as the fuselage and rotor hub; OVERFLOW, a structured overset solver used for evaluating the flow near the blades; and SAMARC, a Cartesian flow solver suited to solving the wake flow. Narducci and Tadghighi used second order central difference schemes to evaluate the fluxes in the near-field flow and fifth order inviscid fluxes for the Cartesian wake solver. A Spalart-Allmaras turbulence model was also employed. The smallest cell sizes were approximately $0.1c_{root}$, and the mesh boundaries were placed $10R$ above the rotor, $30R$ below it, and $20R$ to the side. Whilst there were no absolute performance figures provided, it was seen that the figure of merit in hover had reached a periodic steady state between 4 and 5 rotations [11]. After 5 rotations, adaptive mesh refinement was activated for the background mesh, and a new periodic steady state was reached after 7 total rotations. The convergence to a periodic steady state was much quicker in the forward flight simulations, taking just over 1 revolution after the start of the simulation or any changes to the methods.

Widjaja et al. performed a comparison of the Apache blade aerodynamics with and without the modified British Experimental Rotor Programme (BERP) tip [5]. This analysis was conducted with the CREATE$^{\text{TM}}$-AV Helios v10.0 and Rotorcraft Comprehensive Analysis System (RCAS) software packages. RCAS is a comprehensive modelling tool for rotorcraft systems, featuring a one-dimensional beam theory structural model for the rotor blades and a control system for trimming the rotorcraft [5]. Between the capabilities of RCAS and of the Helios CFD solvers, high accuracy multi-physics simulations can be performed, capturing the effects of phe-

nomena such as aeroelasticity. Widjaja et al. employed an O-grid around each of the rotor blades, resolved to a maximum y^+ value of 1 and totalling 3.6 million grid points per blade. The off body mesh had a Cartesian H-grid topology with a grid spacing of approximately $0.08c$ in the near-wake and reducing grid refinement further from the rotor. This off-body mesh had a total of 36.1 million grid points.

Janakiram et al. performed an analysis of the AH-64A Apache rotor in hover using two lifting surface analysis codes, LSAF and EHPIC, and compared the predictions to experimental data [12]. LSAF is a prescribed wake lifting surface code developed by Kocurek et al. [13]. The prescribed wake model for this code comes from a large collection of experimental wake measurements of model rotors. EHPIC was the other code used in the analysis, a free wake lifting surface code from Continuum Dynamics Inc [14]. Both EHPIC and LSAF codes demonstrated good agreement of integrated loads with the experimental Whirlstand data across the range $0 \leq C_T \leq 0.0012$.

The AH-64 Apache rotor was also simulated in hover and in forward flight by Fitzgibbon et al. [15] and by Barakos [16]. These blade-resolved simulations were performed using the University of Glasgow CFD code Helicopter Multi-Block (HMB3). This code uses the arbitrary Lagrangian-Eulerian formulation for time-dependant domains to solve the Reynolds-averaged Navier-Stokes equations in integral form on a multi-block structured grid [17]. The Navier-Stokes equations are discretised using a cell-centred finite volume approach. A Riemann solver of Osher and Chakravarthy [18] is employed to calculate the convective fluxes, with viscous terms modelled using a second-order central difference scheme. Van Leer's monotone upstream-centred MUSCL scheme [19] is used to provide third-order spatial accuracy. The alternative limiter developed by van Albada et al. [20] is used in regions where steep gradients are encountered, mainly as a result of shock waves. Time stepping is processed with a dual-time-stepping implicit method. Simulations of an isolated, rigid AH-64 rotor were performed at collective angles of 6°, 8°, 10° and 11°. The mesh consists of a blade mesh of 5.4 million cells and a background mesh of 5.3 million cells, totalling 10.7 million cells. HMB3 uses a chimera meshing method to allow for these separately generated grids. Both the hover and forward flight results were accurate over the ranges simulated, although C_Q increased slightly faster with C_T versus the Whirlstand data in hover.

3 Radial Basis Function Mesh Motion

This chapter serves as presentation and review of methods for simulating flows around moving or deforming objects. This is a very wide field with a number of fundamental approaches to this challenge, not to mention the myriad of specific methods within each approach. This chapter begins with an overview and appraisal of these broad approaches, before examining the predominant Arbitrary Lagrangian-Eulerian methods in detail. Within this examination is a detailed discussion of Radial Basis Function methods, which are used in the simulations of this thesis. This inspection includes an overview of their mathematics and early use in CFD, before covering some of the important developments made in the field which have maintained their status as one of the leading mesh deformation techniques.

3.1 Desired Characteristics for Techniques Simulating Flow Around Moving or Deforming Objects

Before introducing just some of the many techniques that can be used to simulate flows around moving or deforming objects, it is appropriate to first consider what is desired from these techniques. Firstly, as a technique for flow simulations, it should minimally degrade the fluid calculations. This is achieved by having as small an effect on the mesh quality as possible. Furthermore, a number of important flow features, especially when considering aerodynamic forces exerted on bodies, are developed in boundary layers. Therefore, it is preferable that the degradation in mesh quality happens far from the object surfaces, with minimal change to the boundary layer. A second consideration is that the simulations should still be practical from a computational standpoint, with the technique having minimal additional calculations. This is particularly pertinent in view of modern, highly refined aerodynamic simulations, with grid sizes growing to many millions or even billions of nodes, as the computational effort would be expected to grow with grid size. Additionally, aspects of the computational cost that may be of interest are the time spent in preprocessing versus per-timestep calculations, and the parallelisation of the methods. Whilst the total time taken over the course of a given simulation is the most important metric, it may often be preferable to spend more of the time in preprocessing than in the calculations required within each step of the simulation. In the case that the simulations need to be run for a longer simulation time than expected, this reduces the total cost. Furthermore, some or all of the preprocessing may be applicable to other simulation scenarios of the same object, thus removing some of the total computations over multiple simulations. The advantages of highly parallel methods are also clear in light of using hundreds or thousands of processors being common in modern CFD. If a method's workload can be distributed across these processors correspondingly, the method will be much more performant than a serial equivalent or code that does not parallelise well.

Secondary desirable characteristics for these methods include high flexibility, simple usage and easy integration with structural models. The flexibility of the methods has two key facets. Firstly, the ability of the technique to be applied to various sim-

ulation scenarios, such as rigid motion, deformation of an object, combined motion and deformation, and combined movement of multiple bodies. Another aspect of flexibility regards the integration of the method with CFD algorithms. This most prominently includes the ability of the method to be applied to both structured and unstructured grids of any cell shape. Additionally, aspects such as not needing additional data, for example mesh connectivity data, improve the flexibility to be easily integrated with different solvers. While a flexible technique is desirable, it should ideally also be simple to use. It is preferable for the method to require minimal user input, whilst working equally well across the range of possible simulation scenarios. If using a method requires trial and error or previous experience to tune method parameters, this will often cost the user a significant amount of time. This is especially true for slower methods used on large grids. Lastly, it is also desirable for methods to be easily integrated with structural models for the purpose of FSI simulations. FSI simulations are a significant use case for these methods, and simplifying the transmission of data between the structural model and fluid simulation can streamline a method's incorporation into a CFD code.

3.2 An Overview of Techniques for the Simulation of Flows Around Moving or Deforming Objects

In order to correctly predict the real-world behaviour of many aeronautical systems, the development of accurate fluid-structure interaction methods is essential. Within this area of aeroelastic simulations, the ability to simulate the fluid flow itself is a significant challenge. One of the primary difficulties is how to deal with interaction between the motion of bodies in the flow and the mesh on which the governing equations are solved. There are two ways to address this issue. One is not having an explicit representation of the bodies in the mesh, in which case the challenge of enforcing the effects of the bodies on the flow must be solved. Immersed Boundary Methods take this route. On the other hand, using traditional body-conforming grids requires an approach to adjust the mesh to motion of the bodies. This can be achieved by using multiple grids, the chimera grid method, or by deforming the mesh points to move with the body, with these methods being a subset of Arbitrary Lagrangian-Eulerian methods.

3.2.1 Chimera Mesh Methods

Chimera grid methods are based on the concept of having separate meshes for each body or section of flow. For example, in the context of rotorcraft simulations, a separate mesh may be used for each rotor blade, the rotor hub and the fuselage, as well as having a background mesh that captures the wake and the flows between the other meshes. Overset grid chimera methods are comfortably the most commonly used chimera method. These techniques have completely distinct grids which overlap each other. There are also patched grid chimera methods, where there are no overlaps between the grids, but one grid maintains its boundaries while effectively cutting through cells on the background grid. As the more popular method, over-

set grids will be discussed first, followed by an evaluation of patched grid chimera schemes.

The primary advantage of chimera grids is that each mesh is able to be generated separately from each other. This significantly simplifies the creation of the meshes and allows each mesh to be optimally shaped and refined, without either being affected by the topology elsewhere in the mesh, or having to carry unnecessary refinement into the far-wake mesh in structured grids [21]. Another decisive advantage of chimera techniques is that they can avoid the usage of the mesh deformation techniques for rigid motions due to the independent nature of the individual grids. This means that there are no concerns regarding degradation in mesh quality from such movements. Combining these aspects also allows for extremely high quality boundary layer meshing with rigid motions. However, when objects change shape, such as in fluid-structure interaction simulations, a mesh deformation algorithm will be required. This means that the simulation will have to run both mesh deformation and chimera grid algorithms, very likely increasing the computational cost over just using a mesh deformation technique.

Another set of disadvantages of chimera methods stems from the interaction of the separate meshes. Firstly, if the overlapping grids are unmodified, there is the potential for multiple grids to simulate flow over the same volume, wasting computational resources. Similarly, background grids may also simulate flow within objects. To avoid this, sections of the major grids, within which minor grids are located, should be removed. This process is called hole-cutting, and should leave a small overlap between grids. This process can be seen in Figure 6, taken from Steger et al. [21], where the dotted points are removed from the primary grid, which is represented by the dashed lines. This hole cutting procedure becomes more complex and expensive when there is relative motion between the grids. Considering that the overlap of the grids will correspondingly change, the hole cutting process has to be repeatedly executed. This incurs an additional computational cost, and requires methods that can robustly and accurately identify which nodes should be removed from the major grids. The data structures must also be updated to identify the new boundary nodes at the hole. Furthermore, points adjacent to these boundaries also must be tracked for the purpose of inter-grid communications.

Another issue stemming from the grid overlap is the treatment of grid boundaries in the overlapping zones, on which boundary conditions should be specified. The major grids have an inner hole boundary, with unknown flow behaviour within the hole, and minor grids are not directly linked with the farfield flow conditions. These boundary values are naturally specified by exchanging flow values between the overlapping grids using an interpolation scheme. There are a number of interpolation methods with various levels of complexity and accuracy. The straightforward n-linear interpolation remains a popular method across many solvers [22], for example in US Army's PUNDIT domain connectivity module [23]. However, many interpolation schemes, including n-linear interpolation schemes, are non-conservative. The impact of non-conservative schemes on the accuracy of the solution has been de-

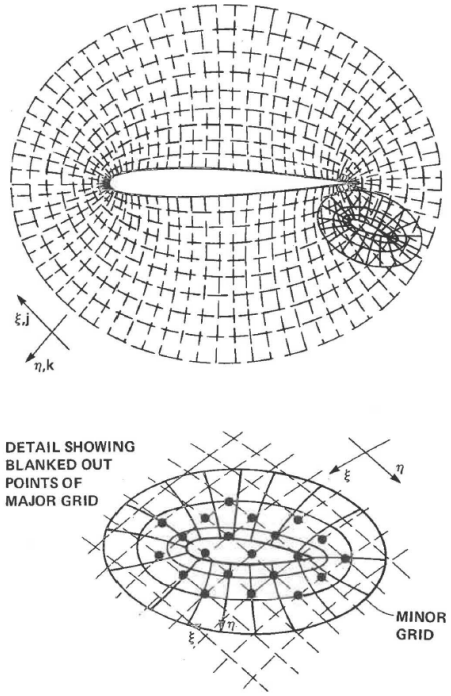


Figure 6: Example of cutting a hole in a major grid using an overset chimera method from Steger et al. [21]

bated within thew chimera grid research [24]. Some claim that these schemes are a non-issue, with solutions converging on an accurate flow prediction with increasing mesh density [24]. This is supported by the results of Meakin, who found that the formal second order solution accuracy was maintained in simulations with tri-linear interpolation, even when there was a shock across the inter-grid boundaries [25]. On the other hand, advocates for conservative methods highlight that conservation is a basic requirement for all schemes and that non-conservative schemes cannot be fully trusted [24, 26]. Additionally, there are also numerical experiments supporting conservation being important to solution accuracy. For example, Davis and Thompson showed that small errors of less than 0.5% in conservation of mass produced errors of about 15% in thrust calculations of a scramjet [27]. Liu and Shyy found that both linear and quadratic interpolation were inferior to conservative boundary treatments in the simulation of lid driven cavity flow [28]. Wang also shows that the conservation error is first order if conservative second order schemes are used on the Chimera grids and there are no discontinuities at the inter-grid boundaries [24]. However, conservative methods generally involve additional computational costs, usually from solving additional systems of equations, or from grid geometry bookkeeping and manipulation.

Patched grid chimera methods avoid the grid overlap issues of the overset grid methods through maintaining the boundary of one grid and cutting through cells on the other grid. Examples of such methods can be seen in Wang [26] and Mohan Rai [29]. A typical diagrammatic example is seen in Figure 7, showing the overlap of grids A and B. The boundary of grid B, Γ_{BO}, cuts through the shaded cells,

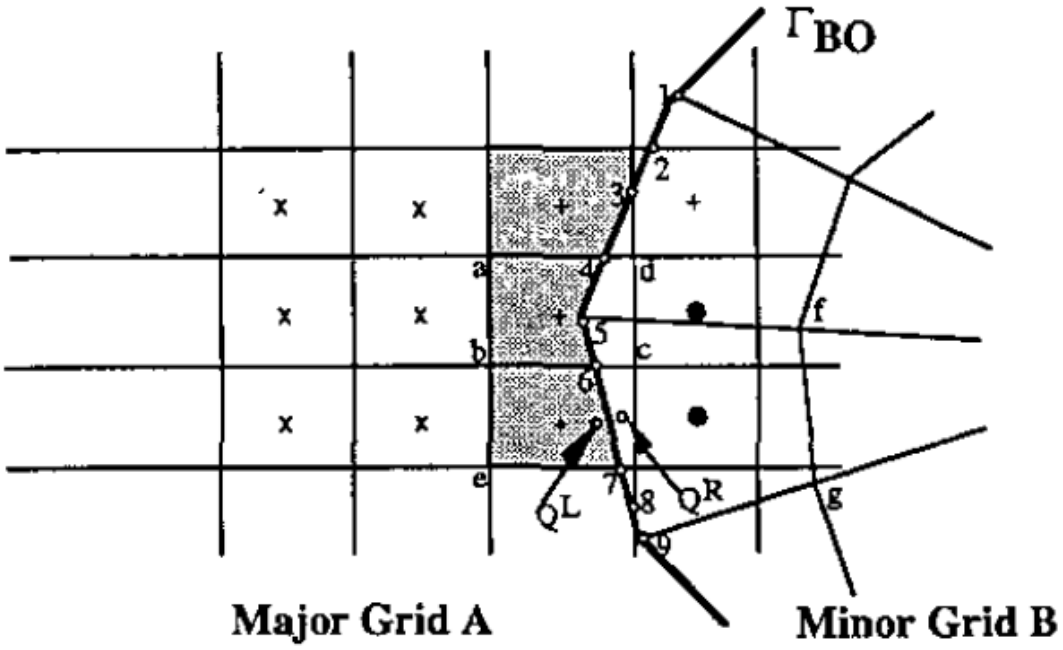


Figure 7: Example of a patched chimera grid method from Wang [26]

amongst others. Wang's patched method calculates the fluxes from grid A along each of the segments of Γ_{BO}, for example segments 12, 23, 34, using reconstruction and Riemann solver stages. The flux across the faces of Γ_{BO} are then calculated by combining the weighted fluxes across each of the segments. These patched methods are generally conservative and avoid interpolation schemes. However, as can be seen in Figure 7, there is a lot of geometric data that needs tracking, which would only become harder for three dimensional simulations. This computational overhead and the acceptable accuracy of overset methods in practice have meant that patched methods have seen limited use in comparison to overset chimera methods.

Whilst chimera methods are mostly simple to use and extremely flexible with regards to the ability to capture complex rigid motions, the primary disadvantage of such methods is the need for a mesh motion algorithm when simulating deforming objects. This also means that a lone chimera method would be nonviable for FSI simulations. Under cases without large relative motions, both the deformation and movement of the bodies could usually be achieved using only the mesh deformation method, which should save computational effort, and the work in implementing two complex modules in the CFD solver. When looking at the solution quality for rigid motions, within each chimera grid the solution should be of the highest quality possible across the methods discussed in this section as there is no mesh deformation, and each mesh can be optimally designed for the flow around each object. This is also true for the boundary layer mesh within each grid. However, with the non-conservativeness of typical inter-grid interpolation schemes, the accuracy of the solution is always inherently somewhat in doubt.

3.2.2 Immersed Boundary Methods

Another set of methods used to simulate flows over moving objects are the Immersed Boundary Methods (IBM). These methods are most often solved on Cartesian grids, with modern implementations additionally taking advantage of Adaptive Mesh Refinement (AMR). Whilst these methods are not new, with the first Immersed Boundary Method detailed by Peskin in 1982 [30], recent advancements have increased their popularity. The use of Cartesian meshes is attractive because their geometrical simplicity allows for the development of efficient, high-order discretisations of governing equations. Furthermore, the process of producing meshes is extremely simple when compared to the generation of boundary-conforming grids. However, it is obvious that for the vast majority of cases, the boundary of the object, around which flow is being simulated, will not conform to the edges of a Cartesian mesh. This raises the question of how to deal with the boundaries. The most straightforward approach is to simply apply the appropriate boundary condition to the Cartesian mesh boundary and refine the mesh in this area, under the premise that smaller cell edge lengths will enable a closer approximation of the boundary. This approach was examined in a study of vortex ring formation in a curvilinear nozzle by Fadlun et al. [31]. The l^2 norm of the axial and radial flow velocities decreased much more slowly than a first-order convergence rate. Whilst this is just one study, it does suggest that this is an ineffective method for dealing with the boundaries. A popular and successful solution to this issue is Immersed Boundary Methods (and the very similar Embedded Boundary Methods). These methods either modify the governing equations with a forcing term that acts locally on the nodes near the immersed boundary (IB), or specify the flow variables at the nodes adjacent to the boundary. The former most often represents the force exerted on the fluid by a solid, whereas the latter more commonly represents physical constraints such as the no slip condition. Additionally, as there is no actual representation of bodies in the mesh itself, there is no explicit need for remeshing or mesh motion, which greatly simplifies the simulation of moving and/or deforming bodies.

However, there are drawbacks associated with the use of IBMs. Regardless of method, it is clear that the boundary mesh nodes will coincide with the IB extremely infrequently. Therefore, the methods that impart these forces and constraints on mesh points or faces introduce another approximation into the solution. Depending on the exact method, the conservation laws may be violated in the affected cells, local order of accuracy may be reduced, and the required grid resolution may be greatly increased [32, 33]. Furthermore, when considering moving IBs specifically, the emergence of nodes from within a body into the flowfield is a notable issue. It is unclear how to calculate derivatives at points which were either within the body on the previous timestep and are not part of the boundary conditions this timestep [34], or points which were enacted upon by a boundary condition on the previous timestep, and whose derivatives would require nodes within a body [32]. Solutions to this issue typically involve further interpolation steps. This highlights another challenge with IBMs: that while the meshing is simple, one must keep track of a number of point groups.This can include the body boundary nodes, which mesh nodes are inside the body, which nodes are being enacted upon by the boundary

conditions, which nodes are being used for the interpolation of the boundary nodes, and which nodes are being used for determining derivatives of points with changing role in the mesh. Furthermore, the cubic cells used by IBMs are the root of significant computational scaling issues when compared to methods with body-conformal grids. The primary issue is of boundary layer resolution. As a body-conformal grid can be refined separately in the wall-normal and wall-parallel directions, the increasingly thin boundary layers at high Reynolds numbers can be compensated for by independently increasing the wall-normal resolution. However, as IBM methods use cubic cells, increasing Reynolds numbers lead to decreased cell lengths in all directions. Mathematically, Verzicco provides approximate correlations between number of cells N and Reynolds number of $N \sim Re^{1.8} \log Re$ for body-conforming grids and $N \sim Re^{2.7}$ for IBM grids [35]. Verzicco concludes that, in spite of the computational operation count per node being at least a magnitude smaller for simple meshes than for curvilinear meshes, the unfavourable cell count scaling makes IBM methods untenable for high Reynolds number flows without additional modelling at walls [35]. These issues have lead to methods with nonuniform Cartesian grids and to Adaptive Mesh Refinement.

The case for using Adaptive Mesh Refinement with IBMs comes from the need for high resolutions, both to simulate high Reynolds number flows and to mitigate errors introduced by the treatment of the boundaries, whilst maintaining a workable cell count. As an example, both Lee et al. [36] and Seo and Mittal [37] found spurious pressure variations in 2D simulations scaling with $\Delta t/\Delta x^2$, with Δt being the timestep size and Δx being the cell edge length. Correspondingly, a reduction in cell edge length and timestep size are necessary to improve IBM solution accuracy. However, when using uniform Cartesian meshes, refinement near the boundary is carried all the way to the mesh boundaries, leading to cells being much smaller than required in the farfield, and hence wasting computational resources. Roma et al. alleviated this problem by using Adaptive Mesh Refinement [38]. This method involves constructing a background Cartesian grid, on which a number of finer sub-grids are located, with each level having a smaller mesh spacing than the level above (typically half or quarter the spacing). The method of Roma et al. refined grids within a specific distance of the IB to resolve the near-body flow more accurately [38], but measures such as vorticity may also be used as refinement criteria to help capture flow features accurately as well. While a nonuniform Cartesian grid with predefined refinement zones could achieve these aims, AMR has some important advantages. Firstly, AMR based on flow properties can adapt to time-varying flows, and hence reduces cell counts when compared to refining the complete volume in which the flow features of interest could occur. More pertinently, a static Cartesian grid cannot account for moving bodies as the surfaces may move outside the refinement zones. Thus, AMR is critical to the feasibility of IBM simulations of flow around moving or deforming bodies, especially at high Reynolds numbers. An example of a Cartesian mesh with AMR can be seen in Figure 8 taken from Capizzano and Cinquegrana [34], depicting the mesh used for the simulation of airflow around an oscillating circular cylinder. Mesh refinement can be seen in the areas near the cylinder surface, resolving the boundary layer, and also in the wake, where the shear layer shedding

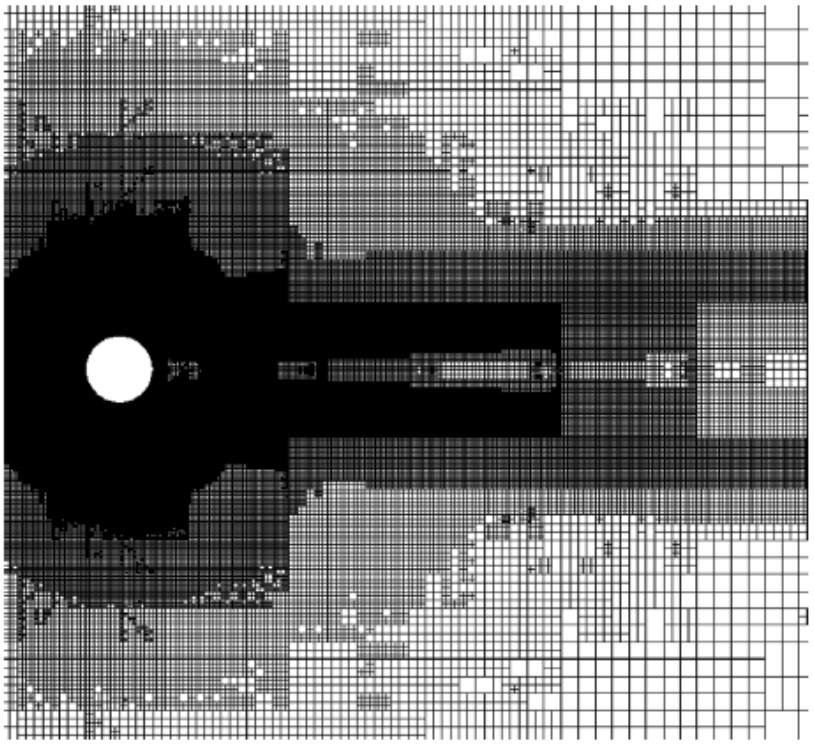


Figure 8: Example of a Cartesian grid IBM using AMR for the flow around an oscillating cylinder, from Capizzano and Cinquegrana [34]

and von Kármán vortex street are found.

While AMR IBM methods enable such simulations, there are also a number of new challenges that it introduces. Having cells of different sizes next to each other requires a method to correctly determine the neighbour-cell flow variables for calculating fluxes and derivatives. Roma et al. achieve this by creating a layer of ghost cells surrounding each sub-grid with the same edge length as the corresponding sub-grid [38]. The flowfield values in these ghost cells are calculated from the actual coincident cells via interpolation. These interpolation schemes must be chosen specifically to maintain the desired order of accuracy of the solution. Roma et al. use quadratic polynomial interpolation schemes to get third order accurate values in the ghost cells, but this is only enough for first-order accurate second derivatives along sub-grid interfaces [38]. A second pertinent issue for AMR is load balancing in parallel computations. Refining cells increases both the number of objects to be stored in memory, and increases the number of operations to be executed on the processor that manages these cells. If the processor cell allocation is static, there is an important possibility that some processors come to manage notably more cells than other processors, and correspondingly slow down the execution of the simulation. Resultingly, a Dynamic Load Balancing (DLB) system is highly advantageous for AMR IBM simulations. Rettenmaier et al. [39] display the advantage of AMR and DLB in three simulations. This included a 2D simulation of capillary rise between two plates, a 3D case of a dam break with an obstacle, and 3D simulation of a car

body. All scenarios showed increased computation speed when using DLB when running parallelised simulations. The AMR simulations were between 1.50 and 2.17 times faster in the capillary case, between 2.13 and 3.86 times faster in the dam burst case, and the car aerodynamics simulation was 1.86 times faster to execute 1957 iterations. AMR is clearly a very useful technique for IBM simulations, but it does add further complications to the simulation code. Code for the cell refinement and coarsening, tracking the variables that determine where to refine the mesh, and interpolation schemes for ghost cell values all add to the complexity. Furthermore, parallel simulations may additionally require load-balancing code to ensure that the simulation is not being bottlenecked by a specific process.

In summary, IBMs are the most flexible methods presented in this section. With the implicit representation of bodies, there are no significant restrictions on the movements or deformations of any number of bodies in the flow. Further, the user input is minimal and high-accuracy simulations can be performed with these methods, especially when using AMR to provide a very fine mesh near the object boundaries. However, these methods do come at a very high computational cost, especially when considering the additional refinement needed to accurately model boundary layer flow when compared to a body conforming grid. IBMs are also not the most practical for integration with existing CFD solvers, considering that AMR and DLB methods would likely have to be additionally implemented in code, as well as the specific discretisation schemes which take advantage of the Cartesian grids.

3.2.3 Arbitrary Lagrangian-Eulerian Methods

Arbitrary Lagrangian-Eulerian (ALE) methods simulate flow on a single boundary-conforming grid. Therefore, as the body within the flow moves, the mesh must also deform or move to correctly capture the flow. As the mesh itself is not a physical object, the mesh velocities may theoretically be chosen arbitrarily. Thus, there are a plethora of approaches to achieving the required mesh deformations. These include the spring analogy, interpolation methods, partial differential equation (PDE) methods, radial basis function methods and the Delaunay Mapping method. These methods will be investigated more thoroughly than the chimera mesh and Immersed Boundary Methods for two reasons. Firstly, these approaches are very common. IBM techniques are still not a common approach and, if not using these, some mesh deformation method is required for deforming objects, even if chimera grids are used. Secondly, the Radial Basis Function methods used for the simulations in this thesis are ALE methods. Therefore, this section enables a thorough comparison with the techniques that are more directly in competition with these RBF methods.

3.2.4 Algebraic Mesh Motion and Regeneration

Interpolation methods are able to be used for mesh motion on grids which are algebraically defined. An example of such a method is the Transfinite Interpolation (TFI) method, which was a popular method for grid generation proposed by Gordon

and Hall [40]. This method allows for a surface or volume mesh to be defined from a set of boundary edges, provided that these edges can be described by a set of equations. The internal mesh structure can be further refined through the specification of edge-to-edge mesh-internal segments, which constrain the internal grid. Factors such as grid spacing are controlled by the choice of "blending functions" [40]. This method was very successful for single-block meshing in particular, with multi-block meshes requiring additional care to deal with the complexities that they introduce. Grid motion for these single block meshes was achieved using two approaches. The first was to simply remesh every timestep, as used by Allen [41, 42, 43], Gaitonde [44], and Gaitonde and Fiddes [45]. This was achievable due to the low computational cost of TFI for small meshes, and had the additional benefit of providing expressions for the mesh velocities and accelerations through the differentiation of the TFI equations with respect to time. The second approach to algebraic mesh motion was to apply displacements to the mesh points according to the displacement of the boundary surfaces. This would take a form similar to:

$$\mathbf{x}(\xi,\eta,\zeta,t) = \mathbf{x}(\xi,\eta,\zeta,0) + \varsigma_1(\zeta)\Delta\mathbf{x}(\xi,\eta,0,t) \tag{7}$$

with

$$\Delta\mathbf{x}(\xi,\eta,0,t) = \mathbf{x}(\xi,\eta,0,t) - \mathbf{x}(\xi,\eta,0,0) \tag{8}$$

where ξ,η,ζ are the curvilinear coordinates, t is time and $\mathbf{x}$ is the grid position [46]. The grid is assumed to have $\zeta = 0$ at the body surface and $\zeta = 1$ at the outer boundaries in this direction. ς_1 is a blending function with $\varsigma_1(0) = 1$ and $\varsigma_1(1) = 0$, controlling how strongly the surface motion affects the mesh at various ζ isosurfaces. This approach is further developed in the work of Allen [46], splitting the motion into a rotation and deformation of a body, allowing for higher grid quality to be maintained under large deformations.

For a large number of applications, especially those with complex geometries, it is impossible to create a single structured grid for the complete simulation domain. This motivated the development of algebraic mesh motion methods which could be used on multiblock meshes. Multiblock meshes require a more complex approach to algebraic grid motion. Firstly, each block may be generated from a different set of functions, requiring some form of piecewise analysis. Secondly, the single block approaches assume a consistent set of curvilinear coordinates. This may not be the case with a multiblock mesh, with mismatching grid coordinates or directions being possible on each connecting face or edge. Dubuc et al. presented a solution to this issue by separating the grid motion into three steps: moving the block corners, moving the block faces, and then moving the block interior [47]. The block corners are kept in position where required, or are displaced with the moving surface. The displacements $d\mathbf{x}$ of the edges are then linearly interpolated according to the displacements of the corner points. Finally, the displacements of the block internal points are calculated using TFI of the face displacements:

$$d\mathbf{x}(\xi,\eta) = \mathbf{f}_1(\xi,\eta) + \Theta_1^0(\eta)\left[d\mathbf{x}_{b1}(\xi) - \mathbf{f}_1(\xi,0)\right] + \Theta_2^0(\eta)\left[d\mathbf{x}_{b3}(\xi) - \mathbf{f}_1(\xi,1)\right] \tag{9}$$

$$\mathbf{f}_1(\xi,\eta) = \varsigma_1^0(\xi)d\mathbf{x}_{b4}(\eta) + \varsigma_2^0(\xi)d\mathbf{x}_{b2}(\eta) \tag{10}$$

where ξ and η are the block local coordinates, $d\mathbf{x}_{b1}$ to $d\mathbf{x}_{b4}$ are the interpolated edge displacements, and Θ_1^0, Θ_2^0, ς_1^0 and ς_2^0 are the blending functions, ς in the ξ direction and Θ in the η direction. This methodology was used to simulate oscillating NACA0012 aerofoils, with rotation about the quarter-chord of up to 4.59° [47]. Dubuc et al. additionally simulate flow around a configuration B Williams aerofoil, with the most demanding case involving flap deflection oscillations of 7°, with a maximal 14° deflection relative to the initial grid. The grid quality was sufficient to extract pressure contours and skin pressure coefficients [47]. Furthermore, this multiblock method was used in more complex scenarios such as the transonic stability analysis of the Goland wing, MDO wing, and Open Source Fighter by Badcock et al. [48], and the aeroelastic study of wind turbines of Carrión et al. [49]. These studies attest to the viability of multiblock TFI, with application to complex meshes and nontrivial amplitudes of motion.

Whilst algebraic mesh motion and regeneration were common methods, the increased complexity of mesh structures, geometries, and especially the popularity of unstructured meshes, has contributed to the development of alternative methods. It is possible to generate multiblock meshes of complicated geometries, as shown by Carrión et al. [49], but finding suitable functions and constraining intervals to describe the surfaces may be a lengthy process involving trial and error. Furthermore, surfaces under large deformations may generate boundary shapes which no longer suit the chosen blending functions.

3.2.5 The Spring Analogy

The spring analogy is a versatile mesh motion method that can be used on both structured and unstructured grids. The spring analogy was first used by Batina, in which each of the unstructured mesh edges is modelled as a spring [50]. The spring stiffnesses are inversely proportional to the length of the edges:

$$k_{ij} = \frac{1}{[(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2]^{\frac{1}{2}}} \tag{11}$$

This means that as two points come closer together, they also experience a larger (fictitious) force pushing them apart, which aims to prevent them from colliding or passing through each other [51]. The x, y and z displacements of each point are then evaluated from the static equilibrium equations [50]. This is commonly achieved through Jacobi iterations.

This method has been validated in simulations with comparison to experimental data in studies of the aerodynamics of the NASA Langley fighter at various angles of attack [50], of the unsteady aerodynamics of an oscillating NACA0012 aerofoil [52], and simulations of a 2 degrees of freedom aerofoil and supersonic flutter of a

flat plate [53].

A widely used improvement to the spring analogy in unstructured 2D grids was published by Farhat et al. [51], in which a set of fictitious torsional springs are introduced which act upon every pair of mesh triangle edges. Farhat et al. suggest a spring stiffness at node i of:

$$C_i^{ijk} = \frac{1}{1-\cos\theta_i^{ijk}} \times \frac{1}{1+\cos\theta_i^{ijk}} = \frac{1}{\sin^2\theta_i^{ijk}} \tag{12}$$

for each triangle with vertices ijk. θ_i^{ijk} is the angle between edges ij and ik. This approach avoids the possibility of zero-area triangles and of edges passing through other triangles, because $C_i^{ijk} \to \infty$ both when $\theta_i^{ijk} \to 0$ and when $\theta_i^{ijk} \to \pi$ [51]. The torsional spring stiffness can also be expressed in terms of the triangle edge lengths l_{ij}, l_{ik} and of the triangle area A_{ijk}:

$$C_i^{ijk} = \frac{l_{ij}^2 l_{ik}^2}{4A_{ijk}^2} \tag{13}$$

Blom suggests that the resistance to cell inversion can be increased by calculating the spring stiffnesses at intermediate iterations, but cautions that this makes the system of equations nonlinear, and hence the iterative evaluation of the mesh vertices could diverge if appropriate precautions are not taken [54]. Farhat et al. suggest using a combination of both linear springs and torsional springs to avoid both vertex collision and inversion of triangles [51]. This can be done by converting the moments exerted by the torsional springs into forces associated with each mesh edge . Figure 9 shows a comparison between the mesh deformation with and without torsional springs from Farhat et al. [51]. In this example, the bottom corners of the mesh are kept in place while the topmost vertex is displaced downwards. It can be seen that the central triangles are "pushed through" the bottom edge of the mesh when only the linear springs are used, while the torsional springs resist this motion as the angles in the bottom triangle approach 0 and π. In a review of the spring analogy by Blom [54], these torsional springs are shown to be practically essential for viscous fluid simulations. Meshes for viscous simulations have high aspect ratio cells near object surfaces to accurately simulate the boundary layers. These cells are particularly at risk for cell inversion or very poor cell quality when using spring analogy methods without torsional springs, including methods modified to have stiffer springs near the moving bodies [54].

Degand and Farhat extended this torsional spring method to three dimensional meshes with tetrahedra [55]. As a vertex can pass through the opposing face of a tetrahedron without creating any zero-area triangles, this method was not as simple as applying the above two dimensional method of Farhat et al. to each face in the mesh. Instead, fictitious triangles are inserted into each tetrahedron, using two tetrahedron vertices and placing the third vertex on the edge connecting the two unused vertices. The forces generated by the deformation of these triangles are then converted into energetically equivalent forces acting upon the four vertices of the corresponding tetrahedron. This method was contrasted to methods only using

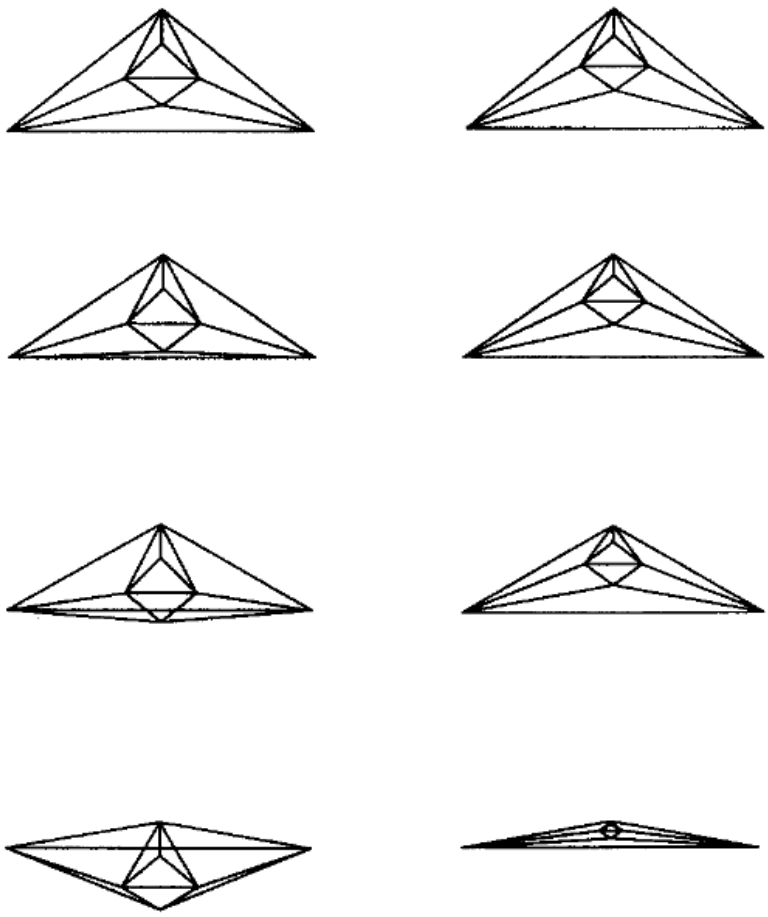


Figure 9: Comparison of mesh motion with linear springs (left), and linear and torsional springs (right) from Farhat et al. [51]

linear springs with simulations of the AGARD wing 445.6 reacting to forced oscillations and to prescribed control surface inputs. Both simulations showed that the torsional spring method was a robust method, with the simulation running to the end-time, whilst the linear spring method produced invalid meshes in both cases after less than a sixth of the total simulation time [55]. In a comparison of the two methods simulating lower amplitude deformation, it was shown that the torsional spring method was approximately 3.5% quicker to simulate up to a specified end-time. This is in spite of the torsional spring method spending 64% more time on the mesh motion, indicating the mesh quality was better, therefore enabling larger timesteps to be taken. Most of this additional time was spent calculating the stiffness matrix. The torsional spring analogy method was also used by Geuzaine et al. in aeroelastic simulations of an F-16, producing accurate results for the first torsional mode frequency and damping at 3.5g and 5g level-flight turns [56]. Lieu and Farhat also used the torsional spring analogy in the aeroelastic simulation of an F-16 [57]. The methodology was sufficient to simulate constant Mach number flight with an excitation applied to the aircraft at high subsonic to low-supersonic Mach numbers.

The spring analogy is quite flexible within the area of ALE algorithms and proven viable for moving and deforming objects in prescribed motion and FSI situations. Furthermore, later developments to these methods, such as those incorporating torsional springs, have enabled acceptable deformation quality for viscous simulations with boundary layer meshing. However, these methods quickly become expensive to solve with large meshes and moving objects, with the algorithms solving a system of equations relating to each grid segment or angle every timestep.

3.2.6 Partial Differential Equation Mesh Motion

Another method of performing mesh deformation is using Partial Differentiable Equations. These formulations solve PDEs defined in terms of the mesh coordinates, with the solutions giving the new coordinates. One motivation for such a method is presented by Löhner and Yang [58]. They propose an example case of a 1D mesh with one stationary end and one moving end. If the velocity of the mesh points varies linearly between the two ends of the mesh, the size ratios of the elements will not change as time progresses [58]. In line with this property of a constant gradient, they suggest finding the mesh velocities from the equation:

$$\nabla \mathbf{k} \cdot \nabla \mathbf{w} = 0 \tag{14}$$

where $\mathbf{w}$ is the mesh velocity and $\mathbf{k}$ is a diffusion coefficient function, which is typically set as a scalar k. Another common form of the PDE is given in Bos et al. [59]:

$$\nabla \cdot (k \nabla \mathbf{x}) = 0 \tag{15}$$

where $\mathbf{x}$ is the displacement field. The value of k can be set to achieve different behaviours from the mesh deformation. For example, $k = 1$ gives a Laplacian velocity smoothing, whereas setting $k \propto 1/h^p, p > 0$, with h being the size of an element, gives a diffusion constant that is smaller for the larger elements of the mesh, leading to higher velocities in this area. As large elements are typically far from the moving boundaries, this may be a desired behaviour.

Many developments have focused on improving this basic PDE method formulation. Löhner and Yang propose a diffusion function of the form:

$$k = k_0 + (1 - k_0) \max\left(0, \min\left(1, \frac{\delta - \delta_l}{\delta_u - \delta_l}\right)\right) \tag{16}$$

which linearly interpolates k between k_0 and 1 between distances δ_l and δ_u from the moving surface [58]. The value of k_0 should be large enough to ensure small velocity gradients close to the moving surface, making the motion there more rigid, while the value $k = 1$ far from the surface should help with uniform deformations away from the surface [58]. Helenbrook advances fourth-order PDEs (as opposed to the hitherto commonly used second-order PDEs) as a way to specify both the

boundary location and spacing [60].

Löhner and Yang demonstrated PDE mesh motion methods with two cases. The first was an unstructured grid of a wing translating upwards and pitching, in which they demonstrate that the modified diffusion coefficient reduces mesh velocity gradients in the vicinity of the wing, reducing cell distortion [58]. Similar results are shown in the simulation of a more complex geometry, namely a hypersonic store release. Baum et al. use the same methodology in their simulations of fuel tank release from an F-16 C/D [61]. This method was sufficient to simulate the full release of the tanks from the aircraft, whilst predicting forces and pitching angles adequately. PDE mesh motion was also used by Dempere-Marco et al. in simulations of cerebral arteries in the context of intracranial aneurysms [62]. They used imaging to determine the motion of the artery walls, which was then imposed on the mesh, with a maximum wall displacement of 3% of the aneurysm diameter.

PDE methods may have the potential to produce very high quality meshes and, as attested to by the studies cited above, can be applied to a variety of scenarios. However, finding the correct set of equations to solve is not a simple process. Fourth order PDEs are required just to have control of the boundary location and spacing, and variable diffusion constants were introduced from early in the research to try to produce higher quality meshes. Furthermore, Rendall and Allen suggest that for large meshes or solvers that require multiple movements of the mesh per timestep, the solution procedure is too computationally costly [63].

3.2.7 Delaunay Mapping Mesh Motion

The Delaunay mapping mesh motion method was proposed by Liu et al. as a computationally quick mesh deformation method that works irrespective of mesh structure [64]. This method begins by generating the Delaunay graph of the points defining the mesh boundaries, or a subset of these points. A Delaunay graph is a tessellation of the convex hull containing a specific set of points [65]. Specifically, each point is connected to the points with which it shares an edge in the Voronoi diagram generated from the complete set. Importantly, this forms a unique tessellation of triangles in two dimensions, and tetrahedra in three dimensions, in the vast majority of cases. For a thorough exploration of Voronoi diagrams and Delaunay tessellation, see the work of Okabe et al. [65]. As the Delaunay tessellation covers the whole mesh, the second step in setting up the mesh motion is assigning each mesh node to one of the tessellation elements. Each node is then described by a combination of relative area or volume coefficients. To illustrate, consider the example in Figure 10 from Liu et al.

Defining the area ratio coefficients e_i as $e_i = S_i/S$, with S being the area of Delaunay triangle ABC and S_i being the area of sub-triangle i, the coordinates of point P can be expressed as:

$$\mathbf{x}_P = \sum_i e_i \mathbf{x}_i \tag{17}$$

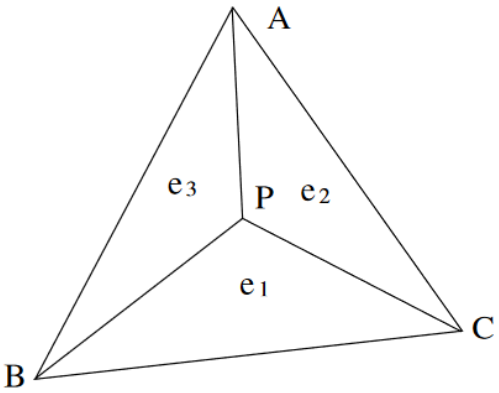


Figure 10: Point P in Delaunay triangle ABC from Liu et al. [64]

where $\mathbf{x}_i$ are the coordinates of node i of the Delaunay triangle. A corresponding formulation for points in Delaunay tetrahedra expresses the point in terms of the volume ratios. When deforming the mesh, the boundary points are first moved as desired. The mesh internal points are then moved according to:

$$\mathbf{x}'_P = \sum_i e_i \mathbf{x}'_i \tag{18}$$

where $'$ denotes the post-deformation values [64]. This movement maintains the area/volume ratios corresponding to each point. This procedure works well for translation and small rotations of the moving boundaries, where the Delaunay tessellation edges do not intersect post-deformation, and hence the area/volume ratios remain positive. Where the deformations are too large, Liu et al. advise dividing the motion in half, and creating a new tessellation for the second part of the movement [64]. This process can be repeatedly applied, dividing the motion into smaller fractions, when the problem persists. However, this method can require a very large number of steps for large rotations, and requires a check of area/volume ratios at each step as well as a retessellation. This could make the algorithm significantly slower, diminishing its primary advantage over other methods, whilst still resulting in a poor quality mesh. An example of this is displayed in Figure 11, showing the rotation of a square by 90° within a larger square mesh, as well as the cell averaged skewness of the resulting meshes. Two mesh motion techniques are shown. On the left is Delaunay mapping mesh motion, and on the right, a radial basis function mesh motion method. The full details of this method are discussed in the following section, but in short, this method calculates the deformation of mesh nodes based on the movement of "base nodes", with the strength of their deformation influence being a function of the distance between the base nodes and the mesh nodes.

The left subplot shows the mesh produced by Delaunay Mapping, achieved by rotating the square by 9° ten times, with retessellation after each step. This is below the explicit threshold for retessellation (producing intersecting edges on the Delaunay Map post rotation), but increased the final mesh quality. Despite this, the maximum node skewness was 16% larger using the Delaunay Mapping Method.

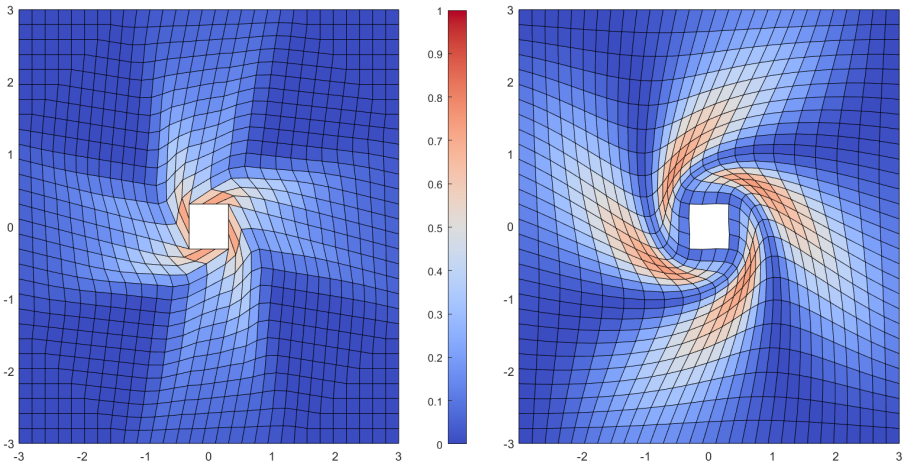


Figure 11: Comparison in skewness between Delaunay mapping (left) and RBF (right) methods

Furthermore, the largest skewnesses can be seen in the area surrounding the square, whereas the maximum values are further from the surface in the RBF mesh. Having the worst mesh quality further from the boundaries is often desirable in aerodynamic studies, as the quality of the boundary layer mesh would be better preserved.

Liu et al. demonstrated their method by performing mesh deformations of a pitching multi-element aerofoil, flap deployment for a multi-element aerofoil, movement and deformation of a sphere in a rigid cube, and the 20° bending of an aircraft wing [64]. The aircraft wing case moves a mesh of 349 038 cells, and involves a comparison to the spring analogy of Batina [50]. Both methods moved the mesh with 20 steps of 1° bending, and the spring analogy meshes were considered converged after a maximum of 50 iterations. The total deformation took 30s for the Delaunay Mapping technique and 312s for the spring analogy. The memory requirements were 114Mb and 72Mb respectively.

This method was further developed by Wang et al. with the Delaunay Graph Mapping and Radial Basis Function (DG-RBF) method [66]. Instead of calculating the displacement of each mesh node according to area or volume ratios as per Equation (18), the node positions are interpolated using radial basis functions (these are explained in detail in Section 3.5 and Section 4.1). In summary, the locations of each of the mesh nodes are controlled by the displacements of the vertices of the Delaunay triangle they are found within, with the influence of each vertex reducing with distance. Wang et al. found that using three points for the RBF interpolations was not sufficient to maintain mesh quality for large rotations, and proposed a different formulation for such motions [66]. This formulation interpolates the angular displacement as opposed to linear displacement, which is then converted back into a linear displacement for the calculation of the new node locations. This approach was tested against the standard Delaunay Mapping method over a range of test cases,

including rotating and translating blocks, a NACA0012 aerofoil, a deforming sphere, and the bending of an aircraft wing. Whilst the DG-RBF method was approximately 1.5 times slower than the Delaunay Mapping method, this is still very quick when compared to other mesh motion algorithms, being orders of magnitude faster than the RBF method of de Boer [67]. The DG-RBF method was clearly superior to the standard Delaunay Mapping method with regards to mesh quality. The size-skew metric minimums were higher across all test cases (with higher values being desirable), as well as the average values being higher or equivalent. Furthermore, the areas of worst cell quality were consistently further from the moving boundaries. However, the average cell quality and boundary layer cell quality were still inferior to a pure RBF method.

While Delaunay Mapping techniques are very low cost, the low deformation quality and robustness to large motions and deformations severely limits their usefulness. In addition, the nature of the mapping provides little ability to preserve the grid quality in the boundary layer. Whilst these issues are somewhat alleviated by the DG-RBF methods, the grid quality, particularly in the boundary layer, was still worse when compared to RBFs.

3.3 Radial Basis Function Mesh Motion

Radial Basis Function mesh motion is another method under the category of Arbitrary Lagrangian-Eulerian methods. These methods were first used for mesh motion by de Boer et al. [67], but have a background in interpolation problems. In the most concise terms, these methods deform the mesh by using a set of nodes to control the motion, whose positions are prescribed or determined by an external calculation. The influence of these nodes on the surrounding mesh points is a function of a norm of the difference between the two points: $F(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{x}_0\|)$. Typically this norm is the Euclidean distance, and the function ϕ is greatest at the controlling nodes and decays to zero. The full method will be covered in detail in Chapter 4.1, but a few terms must be introduced now in order to adequately explore the literature on this topic.

For the basic implementation of RBF mesh motion, the influence of the nodes controlling the motion, henceforth called the base set, at any point is specified as the sum of the RBF influences from each point in the base set multiplied by corresponding constants α_j:

$$F(\mathbf{x}) = \sum_{j=1}^{N_b} \alpha_j \phi(\|\mathbf{x} - \mathbf{x}_j\|) \tag{19}$$

where N_b is the number of nodes in the base set and ϕ is the chosen RBF. This total influence can be used to define the coordinates of points in the mesh. This function should return the location of any point in the base set when it is the input to the function, since their coordinates are known or specified. This can be expressed mathematically as:

$$\mathbf{B} = \mathbf{\Phi}\boldsymbol{\alpha} \tag{20}$$

where $\mathbf{B}$ is a vector of the base node coordinates, $\boldsymbol{\alpha}$ is a vector of the constants, and $\mathbf{\Phi}$ contains the RBF evaluations for every combination of base nodes. Each element of a row of $\mathbf{\Phi}$ represents a corresponding term of Equation (19) without α_j. Lastly, the coordinates of the other nodes are calculated from the locations of the base set, $\mathbf{\Phi}$, and the base set to volume nodes influence matrix $\mathbf{\Psi}$:

$$\mathbf{V} = \mathbf{\Psi}\mathbf{\Phi}^{-1}\mathbf{B} = \mathbf{H}\mathbf{B} \tag{21}$$

Similarly to $\mathbf{\Phi}$, $\mathbf{\Psi}$ contains the evaluation of the RBF for each combination of nodes in the base set and volume mesh. Equation (21) is often evaluated in terms of a difference in coordinates, calculating $\Delta\mathbf{V}$ from $\Delta\mathbf{B}$. Within this implementation there are two principal elements which affect the geometry of the deformed mesh: the choice of RBF and the choice of base set nodes.

From the overview of the mathematics of RBF mesh motion, it can be seen that this method only operates on the nodes in the mesh. This means that RBF methods are largely agnostic to the mesh structure. Importantly, they are equally applicable to structured grids and unstructured grids of any cell geometry. Similarly, operating only on nodes means that no mesh connectivity data is necessary for the use of these methods. Furthermore, the mesh cell geometry has no explicit influence in the deformation calculations unlike, for example, the spring analogy.

3.4 Selection of Radial Basis Functions

The selection of the radial basis function ϕ can have a significant impact on many aspects of the mesh deformation. From an intuitive viewpoint, it determines how the influence of each base node varies over space, and hence the shape of the deformed mesh. Mathematically, it determines the structure of the $\mathbf{\Phi}$ and $\mathbf{\Psi}$ matrices, and hence also the properties of the inverse $\mathbf{\Phi}^{-1}$ and which methods are suitable for the inversion.

Radial basis functions may be divided into two groups with distinct properties: functions with global support and functions with compact support [67]. Functions with global support may be nonzero at all values, whereas functions with compact support maintain the relationship:

$$\phi(\xi) = \begin{cases} f(\xi) & 0 \leq \xi \leq 1 \\ 0 & \xi > 1 \end{cases} \tag{22}$$

where $f(\xi)$ is a non-negative function [67]. The cutoff value of $\xi = 1$ is chosen to simplify the general mathematics of RBF generation, and correspondingly may not suit every implementation case. Therefore, ξ is typically defined as $\xi = \|\mathbf{x} - \mathbf{x}_0\|/r$, where $\mathbf{x}_0$ is the RBF centre, and r is a quantity known as the support radius. With this definition, the values of compact support RBFs decay to 0 at r, which can be

Table 2: Example radial basis functions

Name	**Definition**
Wendland's C0	$(1-\xi)^2$
Wendland's C2	$(1-\xi)^4(4\xi+1)$
Wendland's C4	$(1-\xi)^6(35\xi^2+18\xi+3)$
Wendland's C6	$(1-\xi)^8(32\xi^3+25\xi^2+8\xi+1)$
Euclid's Hat	$\pi(\frac{1}{12}\xi^3 - R^2\xi + \frac{4}{3}R^3)$
Thin Plate Spline	$\xi^2 \ln(\xi)$
Gaussian	$e^{-c\xi}$
Hardy's Multiquadratic	$\sqrt{c^2+x^2}$
Hardy's Inverse Multiquadratic	$\frac{1}{\sqrt{c^2+x^2}}$

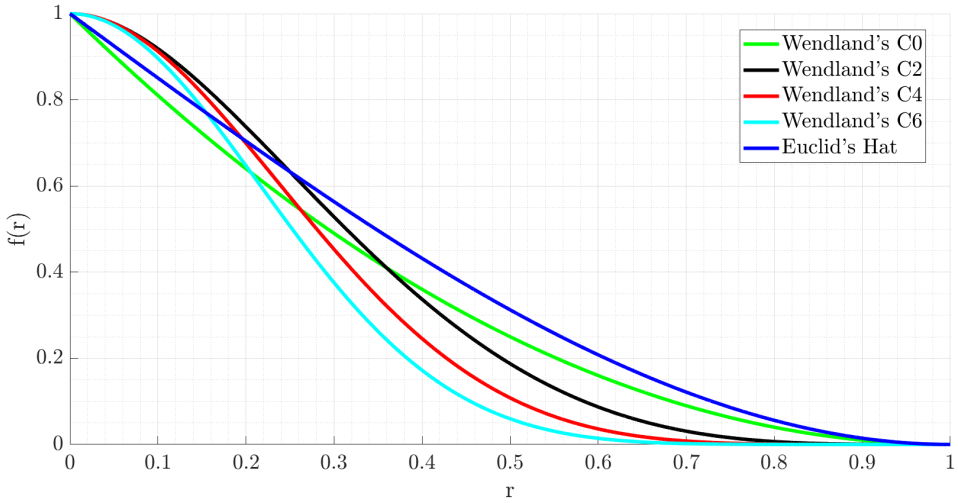


Figure 12: Graphs of example radial basis functions with compact support

chosen to suit the given application. Table 2 shows a number of functions that may be used as RBFs [67, 68, 69]. In Table 2, c refers to an arbitrary constant and R used in the Euclid's hat function is defined to be half the desired support radius. The Wendland functions shown are those with desirable characteristics on $\mathbb{R}^3$. Wendland's functions and the Euclid's hat function are typically used as compact support functions, while the other functions are typically used as global support functions. The RBFs with compact support are plotted in Figure 12, and RBFs with global support in Figure 13. Of note in Figure 13 is that some functions grow with increasing r, whereas others decay. A less commonly used naming scheme calls these functions global functions and local functions respectively.

Another desirable property of an RBF is that it is a positive definite function. A function ϕ is positive definite if it is continuous, even, and if:

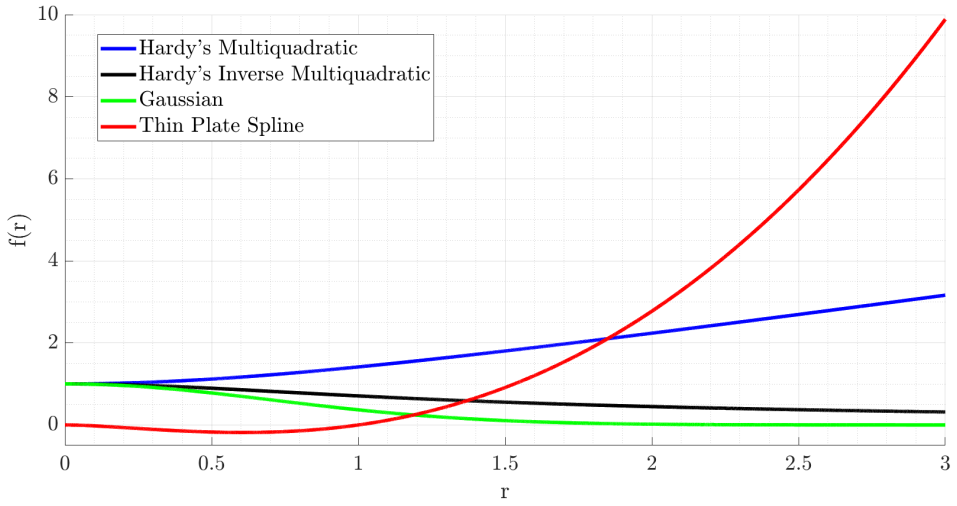


Figure 13: Graphs of example radial basis functions with global support

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\alpha_i\alpha_j\phi(x_i - x_j) > 0 \tag{23}$$

for all $N \in \mathbb{N}$, all $\alpha \in \mathbb{R}^N \setminus \{0\}$ and all pairwise distinct $\{x_1, \ldots, x_N\} \subseteq \mathbb{R}^d$ [70]. Positive definite functions have the advantage of ensuring that the interpolation problem of matching the base set locations, solving Equation (20) for $\boldsymbol{\alpha}$, has a unique solution [71]. In turn, this guarantees that $\boldsymbol{\Phi}$ is invertible [69]. Functions with compact support are also advantageous due to the matrix structures they generate. Because these RBFs evaluate to zero for inputs greater than the support radius, a targeted selection of the support radius can ensure that the RBF is zero for a large number of point combinations. This means that $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ will have more sparse structures, which can avoid the use of matrix preconditioning or specific evaluation techniques that are required for the fuller matrices generated by globally supported functions [69]. The Wendland's RBFs are noted for being the lowest degree polynomials that are C^n (have a continuous nth derivative) basis functions [69].

However, not all radial basis functions are positive definite. In this case, a polynomial term $p(\mathbf{x})$ can be added to Equation (19) to achieve the same properties. The coefficients of the polynomial can be evaluated through the requirement:

$$\sum_{i=1}^{N_b}\alpha_i q(\mathbf{x}_i) = 0 \tag{24}$$

for all polynomials $q(\mathbf{x})$ of degree lower than or equal to that of $p(\mathbf{x})$ [67]. With this polynomial term, RBFs can be classified as conditionally positive definite of order m if they satisfy Equation (23), with α that satisfy Equation (24) for all $q(\mathbf{x})$ of degree $\leq m$ [72]. Conditionally positive definite functions with appropriate polynomials also produce unique interpolation solutions, like positive definite functions do [67]. With regards to choosing the polynomial, Murray makes the observation

that conditionally positive functions of degree m are also conditionally positive of degrees $n > m$ [72]. This means that the degree of the polynomial is only capped by the number of base points that can be interpolated from, as the polynomial coefficients must be determined by these.

There are, additionally, other practical reasons to include a polynomial term in the calculation of the point to point influences. A set of three papers by Flyer et al. and Bayona et al. describe the benefits of the use of the polynomial terms in RBF finite difference approximations [73, 74, 75]. They demonstrated that adding polynomial terms to Gaussian or polyharmonic spline RBFs eliminates the issue of stagnation error, where increasing the refinement of the interpolation grid does not increase the accuracy of the approximation. They also attest to the polynomial term being able to mitigate issues such as poor system condition while maintaining high order convergence. Murray also advances the polynomial term as helpful for mesh deformation from simple body motions, such as translations [72].

Aside from the theory of RBF choice, applied analyses of RBFs have also been performed. Boer et al. analysed a number of compact and non-compact RBFs in an example application with regard to the minimum of the size-skew metric (values closer to 0 being worse), the mesh average of this metric, as well as the effect of r on the mesh quality and computing time. Their analysis concluded with recommending the Wedland's C2 function for both high mesh quality and computational efficiency [67]. Rendall and Allen also performed an analysis to evaluate RBFs [76]. According to their analysis, the change in orthogonality at a certain point is proportional to the derivative of the RBF at that point. Considering that preserving the orthogonality in boundary layers is desirable, they recommended RBFs with a zero derivative at $r = 0$. They also state that higher order functions tend to degrade the condition of the system of equations, and hence recommended the Wedland's C2 function as the lowest order function with the aforementioned zero derivative property out of the functions that they surveyed. Wedland's C2 function has also been used in numerable studies, such as those from Rendall et al. [63, 68, 76, 77], where it was selected for its balance of mesh motion quality and matrix conditioning, Kedward et al. [2], Estruch et al. [78], Murray [72], Wang et al. [66] and was given a favourable appraisal by Beckert and Wendland [79] and by de Boer et al. [67].

3.5 Full Base Set Radial Basis Function Methods

One of the other crucial decisions in implementing RBF mesh motion is the selection of the base set. Whilst selecting all points on the surface of an object is the most simple solution to the problem of point selection, it is also in many aspects the most ideal. Selecting all surface points has the obvious advantage of specifying the exact location of the complete surface. This is important when performing simulations where aerodynamics is a primary focus. For example, in the case of this thesis, inaccurately specifying the shape of the rotor blade could lead to aberrations in the simulation of airflow over a particular section, which in turn would produce inaccu-

rate load predictions. Furthermore, selecting the complete surface has the simplest implementation. Surfaces are most often easily identifiable in the mesh data and there are no additional calculations required, as opposed to methods discussed later. Given the simplicity and ideality of this method, it has been popular, particularly in the earlier research employing RBF mesh motion [63, 67, 78, 59, 80].

The primary disadvantage of this method is that the scaling of its mathematics is unfavourable. Firstly, the required computing power grows quickly with increasing size of the base set. Solving the systems of equations takes $O(N_b^3)$ operations, and updating the mesh requires $O(N_b N_v)$ operations, where N_b is the number of base nodes and N_v is the number of volume nodes in the mesh [2]. Additionally, de Boer et al. found that the computing time required to deform a mesh scaled faster than linearly when the number of base nodes was increased, while keeping the number of volume nodes constant [67]. This scaling becomes problematic in application to modern 3D meshes, where the surface grids may contain many thousands or millions of points. The other major effect of increasing N_b is the ill-conditioning of the systems of equations, particularly noticeable in the deteriorating condition numbers of $\mathbf{H}$ [72] and $\mathbf{\Phi}^{-1}$. In less critical cases this may lead to lowered mesh quality, as less smooth RBFs may need to be chosen in order to mitigate the increase in condition number [76]. However, the number of base nodes may reach a point where numerical artefacts are inevitable in complex applications [72]. As a result of these pressures, new methods were required to apply RBF mesh deformation to meshes of increasing size and complexity.

3.6 Methods with Reduced Base Sets

Given the impracticality of using all the aerodynamic surface points as base nodes on meshes suited to aerodynamic or aeroacoustic Navier-Stokes simulations, the exploration of reduced base sets was an natural route of investigation. Using only a reduced base set comes with a notable drawback: only the base nodes are guaranteed to move to the desired final location, as all other points will be moved by the RBFs. This is an obvious issue when the shape of the deformed objects are important, for example, in aerodynamic studies. There may also be impacts on boundary conditions when the relevant boundaries are within the support radius of the base set. In this case, when one cannot add every boundary point to the base set, there will be movement of the boundaries. This may have notable effects on boundary conditions such as periodic or wall BCs.

Considering these surface position errors, the simplest approach would be to decide on an acceptable level of error, and to choose a number of base nodes so that the error at each point on the surfaces and/or boundaries are below this threshold. The primary issue with this approach is that the reduction in error from adding support nodes drops off almost exponentially. As an example, Figure 14 from Rendall and Allen [76] shows the error in the surface deformation of an ONERA 7A rotor as the number of support nodes is increased.

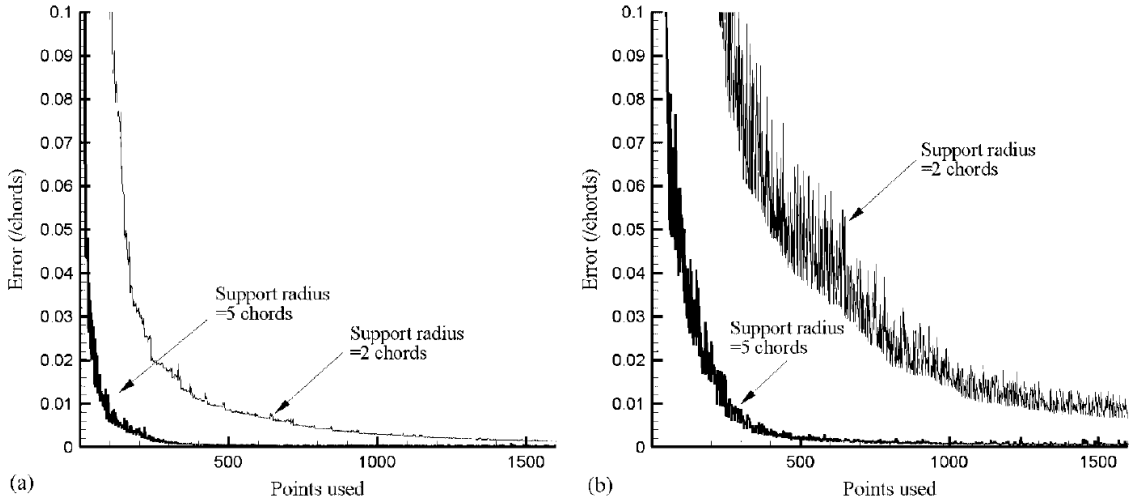


Figure 14: (a) average error and (b) maximum surface deformation error of an ONERA 7A rotor [76]

As a result of the diminishing returns and increasing computational costs, a number of methods were introduced to maintain exact locations of specific nodes when using reduced base sets. Rendall and Allen introduced an additional error correction step in their simulation of the ONERA 7A rotor [76]. The total motion for each node is described as:

$$\Delta_f(\mathbf{x}) = \Delta_r(\mathbf{x}) + \Delta_c(\mathbf{x}) \tag{25}$$

where Δ_f is the total displacement, Δ_r is the displacement from the RBF movement, and Δ_c is the displacement from the correction step. Δ_c is calculated using:

$$\Delta_c(\mathbf{x}_e) = \Delta_n \phi\left(\frac{\|\mathbf{x}_e - \mathbf{x}_n\|}{r}\right) \tag{26}$$

where r is the support radius, point e is the point being moved and point n is the nearest surface point. Δ_n is the difference between the location the nearest surface point is moved to by the RBFs and where its exact location should be:

$$\Delta_n(\mathbf{x}_n) = \Delta_a(\mathbf{x}_n) - \Delta_r(\mathbf{x}_n) \tag{27}$$

This method can be seen as a local, single point RBF-interpolation correction to the surface and near-field [76].

In a study of RBF-based FSI in turbomachinery, Rozenberg et al. maintain the location of periodic boundaries using a modified norm as input to their chosen RBF [81]. Instead of the most commonly used Euclidean distance, they chose a norm of the form:

$$d = \sqrt{d_x^2 + d_y^2 + d_z^2} \tag{28}$$

where:

$$d_y = \frac{\lambda}{\pi} \sin\left[(y - y_{ci})\frac{\pi}{\lambda}\right] \tag{29}$$

assuming that the periodic boundaries are in the y direction and that the associate pitch is λ.

Another method of preserving mesh boundary shapes specifically is shown in the work of Jakobsson and Amoignon [82]. Jakobsson and Amoignon modify the displacements generated by the RBFs to make sure that they decay to zero at a specified distance from the object being moved, which is chosen to be closer than the mesh boundaries. Jakobsson and Amoignon define a modulation function:

$$\tilde{\omega}(t) = \begin{cases} 1, & t \leq 0 \\ 1 - t^2(3 - 2t), & 0 \leq t \leq 1 \\ 0, & t \geq 1 \end{cases} \tag{30}$$

And then define the cutoff function as:

$$\omega(\mathbf{x}) = \tilde{\omega}\left(\frac{\|\mathbf{x} - \mathbf{x}_{\text{mean}}\| - R_1}{R_2 - R_1}\right) \tag{31}$$

where R_1 and R_2 are positive numbers and $\mathbf{x}_{mean}$ is the coordinates of the chosen centre of the cutoff function. This is then applied to the displacement of the mesh, modifying Equation (21) to be equivalent to:

$$\Delta\mathbf{V} = \omega\mathbf{H}\Delta\mathbf{B} \tag{32}$$

In this case, R_1 can be interpreted as an inner radius, where points within this distance of $\mathbf{x}_{mean}$ are unaffected by the cutoff function. Similarly, R_2 can be interpreted as an outer radius, with points outside this radius being unmoved.

3.7 Greedy Algorithms in Radial Basis Function Mesh Motion

When using RBF methods with reduced base sets, the important question of which points should be used to form the base set must be considered. This has a number of answers, ranging from mathematically motivated selection algorithms to experientially validated, arbitrary point selection. Greedy algorithms are one of these mathematically motivated methods, that has been used in application to both mesh deformation and interpolation problems. The following description presents an applied view of these methods, but a thorough mathematical description and analysis can be found in the works of Schaback and Wendland [83, 84].

The basic premise of greedy point selection algorithms is that points are added iteratively to a reduced base set $\mathbf{x}_{\text{red}}$ until a certain condition is met. The points selected are those with the maximum value of some criterion (hence the algorithm

being "greedy"). The pool of candidate points that can be added to $\mathbf{x}_{\text{red}}$ should be the set of points that would make up the complete base set. This set will be represented as $\mathbf{x}_{\text{can}}$. As previously mentioned, this generally corresponds to the surface points of the moving objects. This pool is also the set of points over which the criterion is evaluated every iteration. There are many criteria that can be used for point selection. These include the maximum interpolation error [83], or analogously for mesh motion, the maximum distance between nodes moved by the RBFs and their desired locations [68]; the maximum of a power function as defined by De Marchi [85]; and the maximum distance to the other base set nodes [85]. This thesis uses the position error as the selection criterion, as it has been shown effective in Rendall and Allen [68]. Intuitively, this also measures the error between the desired surface and the surface generated with the RBFs at each point, which will therefore be minimised at each step of the algorithm. The iteration loop of a greedy RBF point selection algorithm can be described in more detail with the following steps:

1. Calculate the exact displaced locations of $\mathbf{x}_{\text{can}}$: $\mathbf{x}_{\text{ex}}$
2. Solve the RBF system with the current $\mathbf{x}_{\text{red}}$
3. Evaluate the new positions of $\mathbf{x}_{\text{can}}$ from the RBFs: $\mathbf{x}_{\text{rbf}}$
4. Evaluate the error at each point $\mathbf{E} = \|\mathbf{x}_{\text{rbf}} - \mathbf{x}_{\text{ex}}\|$, with $\|\cdot\|$ being the Euclidean norm
5. Find the maximum error $E_{\max}$
6. Check the stop criterion. Stop if the criterion is met, continue otherwise.
7. Find the candidate point where this maximum error occurs: $\mathbf{E}(\mathbf{x}_{\text{can}_i}) = E_{\max}$
8. Add $\mathbf{x}_{\text{can}_i}$ to $\mathbf{x}_{\text{red}}$ and, depending on the implementation, remove $\mathbf{x}_{\text{can}_i}$ from $\mathbf{x}_{\text{can}}$
9. Repeat from step 1.

The ideal location of the check depends on the choice of this criterion. A common stop criterion is a desired maximum surface error, as used in [3, 86]. Using this criterion, the algorithm must complete step 5, and hence the earliest and optimal location for this check is between steps 5 and 6. Other possible criteria include an average surface error and choosing an arbitrary number of points that the reduced base set should have. In the latter case, the algorithm loop can be implemented as a for loop rather than a while loop, and hence no explicit check of this criterion is necessary.

In the context of interpolating functions with one-dimensional output, Schaback and Wendland proved that a greedy RBF algorithm, which adds the node with maximum error at each iteration, converges linearly to the interpolation function that uses all points where the original function is defined [83]. Rendall and Allen display

the error at each iteration of a greedy algorithm in application to the deformation of Brite-Euram MDO aircraft wing [68]. This is shown in Figure 15, where it can be seen that the error approximately follows an exponential decay. This gives some credence to thinking that the rate of convergence in more complex situations is also close to linear.

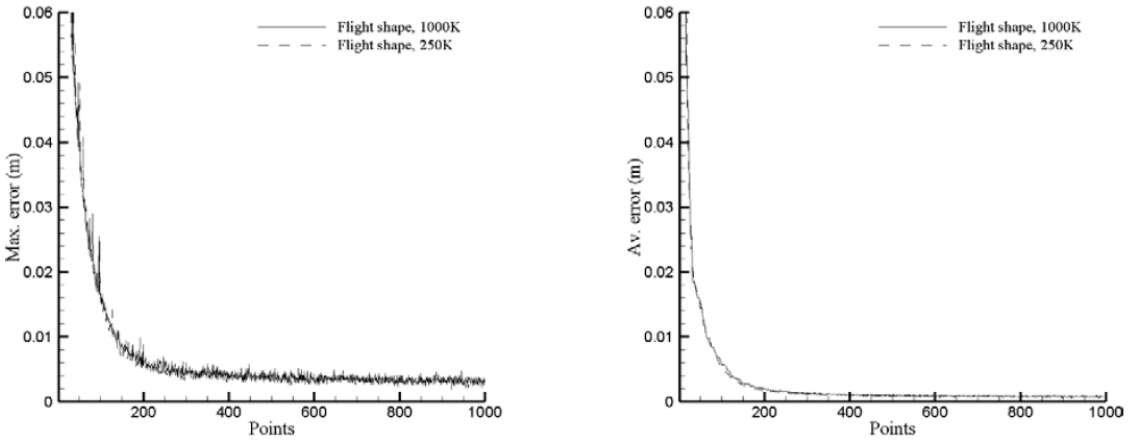


Figure 15: Maximum and average error of a wing mesh deformed to its flight shape with n points in the reduced base set [68]

Whilst the convergence of greedy methods has been shown to be satisfactory, the rate of convergence can be an issue when considering that the iterations will not be computationally fast for large candidate surfaces. As $\mathbf{x}_{\text{red}}$ grows, constructing and multiplying by $\boldsymbol{\Psi}$ requires many more operations, as does inverting $\boldsymbol{\Phi}$. For this reason, there are modified greedy RBF methods that reduce the time taken per iteration. Rendall and Allen propose a system where the full RBF system is not solved every iteration. Instead, a correction factor is calculated at the location of maximum error. This factor is added to the value of the $\boldsymbol{\alpha}$ vector corresponding to the RBF centred on the maximum error node, and this node added to the reduced base set if it is not already in it. $\mathbf{x}_{\text{rbf}}$ is then updated by adding the influence corresponding to the correction factor, rather than re-solving the whole system. Whilst this method is much faster than solving the RBF system every iteration, the error reduction is not optimal. Considering the RBF system is only solved once, the greedy algorithm is selecting points based on an approximation of the surface deformation at each iteration. This leads to consistently larger errors in the final deformed surface shape compared to the standard greedy algorithm after the same number of iterations [68]. Therefore, Rendall and Allen also propose a hybrid method, where the iterations normally use the correction factor method, but there is a full solve of the RBF system every n iterations. This method shows much improved error values compared to the correction factor method, even with a full solve only every 40 iterations. This hybrid algorithm approaches the error values of the standard greedy algorithm the more frequently the full RBF systems solves are performed. Rendall and Allen also perform a comparison in computing speed between the three algorithms, with the hybrid algorithm employing a full solve every 10 iterations. Using a test case with 11 593 nodes in the surface mesh, the time spent solving the RBF systems with

either of the new algorithms was approximately a quarter of the standard greedy algorithm when selecting 200 points. When 400 points were selected, this decreased to just under a sixth. The points selected by the correction factor method were seen to be more clustered in comparison to the standard algorithm, but this was largely alleviated by the hybrid algorithm.

Li et al. take an alternative solution to increasing the speed of the greedy algorithm. Considering the majority of the time is spent on solving the RBF system and updating the candidate point positions, the execution time could be decreased by adding multiple points to the reduced base set per iteration [86]. However, the specifics of this selection must be considered, as simply adding the n points with the highest error generally leads to selecting clusters of candidate points that are neighbours [86]. Adding these clustered points only reduces the surface error in the local region and, resultingly, this does not do much to increase the convergence rate over the whole surface when compared to selecting one node per iteration. Li et al. addressed this by first assembling a list of surface neighbours for each surface point, and then only selecting points which have the largest error of all its neighbours [86]. Li et al. also presented two different methodologies for choosing how many nodes to add to the reduced base set per iteration. The first of these is simply adding n points per iteration. However, this method may be nonoptimal when n is large, due to the algorithm adding points which don't correspond to large scale features of the surface error [86]. The second method is an adaptive selection strategy based on the rate of error reduction per iteration. In the early iterations, the number of nodes in the reduced base set is small. Therefore, solving the RBF system is quick, and so are the early iterations. In this circumstance, there is no need to select multiple points per iteration [86]. In contrast, at later iterations the RBF system takes much longer to solve, and picking multiple nodes per iteration provides more benefit. In implementation of the adaptive method, the error reduction gradient is calculated as the percentage reduction in error value between a given iteration and the previous iteration. The number of points to be selected is calculated from this gradient, with fewer points being selected the steeper the gradient. The user must also specify cut-off values: a maximum gradient, above which only one point will be selected in the current iteration, and a minimum gradient, below which a maximum number of points are selected. These two methods were compared to a standard greedy implementation in application to the test case of a swimming fish. The fish had a geometry as defined by Kern and Koumoutsakos [87], and the deformation of the body was sinusoidal along its length. Two surface error stop criteria were applied, $5 \cdot 10^{-4}$ and $1 \cdot 10^{-4}$. Across both surface error criteria there was less than 4% difference in the time taken for updating the volume nodes, with the time taken being proportional to the number of reduced base nodes. However, there was a significant difference in the time taken by the greedy algorithms. In the case with $5 \cdot 10^{-4}$ maximum error, selecting 3 points per iteration reduced the runtime by 64%, and the adaptive algorithm reduced it by 78%. For the case with a $1 \cdot 10^{-4}$ error criterion, choosing 10 points per greedy iteration reduced the execution time by 89% and the adaptive algorithm reduced it by 86%. The increased savings are due to there being more points selected, and hence the difference in number of iterations

is greater.

3.8 The Multiscale Radial Basis Function Method

The multiscale method was developed by Kedward et al. as a fast method for RBF mesh motion which also provides exact movement of moving surfaces, without the separate error correction steps or RBF modifications discussed in Section 3.6 [2]. The multiscale method bears resemblance to the multistep RBF method for scattered data interpolation of Floater and Iske [71]. This method separates the original dataset into a number of nested datasets, which should have evenly distributed points and which increase in point density. Each subset is assigned a support radius that decreases with the increasing density of the points. In doing so, the general large scale behaviour should be captured by the less dense subsets, which propagate their influence over a wide area with large support radii [71]. The smaller subsets then interpolate the finer details of the data with the smaller support radii. Floater and Iske compared the multistep method to the traditional one step method as described in their paper [71]. The "one step method" (as discussed previously) could trade off increased computing time for lower interpolation error, but generally failed to achieve the simultaneous speed and accuracy of the multistep method [71]. As an example, they state that with a specific support radius the one step method produced errors over 100 times as great as the multistep method, but also took ten times longer.

Similarly to the multistep method, the multiscale method aims to capture the gross mesh motion with a small selection of primary deformation mapping points, which have a large and constant support radius. The method specifies the location of the remaining surface points to keep the surface movements exact, and additionally uses them to propagate smaller scale refinements of the mesh motion. Importantly, each stage of the refinement is chosen so that it has no influence on all proceeding refinement nodes nor on the base set [2]. This allows the resulting system of equations to have a simple and fast solution procedure.

The multiscale method begins by selecting a subset of the boundary mesh as the base set, which acts analogously to the base set of the "standard" RBF methods. The remaining surface points are used as refinement nodes. The refinement nodes distinguish themselves from the base set in that they do not influence the base set, nor any refinement points in the previous refinement levels. Correspondingly, the base set will typically have a constant, large and arbitrary support radius, whereas the support radii of the refinement nodes are deterministically chosen and dependant on the refinement level and the mesh. The algorithm for assigning the refinement node support radii is based on the measure of separation distance. The separation distance of a dataset is defined as the radius of the largest sphere that is centred on one point of the dataset and contains no other point. Where Ω is this set of points and Ξ is the separation distance, this can be expressed mathematically as:

$$\Xi = \max_{i \in \Omega} \min_{j \in \Omega, i \neq j} \|\mathbf{x}_i - \mathbf{x}_j\| \tag{33}$$

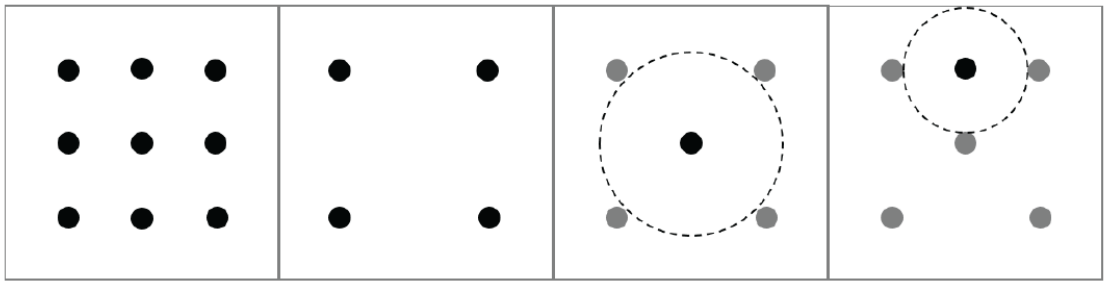
where $\|\cdot\|$ is the Euclidean distance. Correspondingly, the separation distance of a specific point within Ω is the radius of the sphere centred on this point which contains no other points:

$$\Xi_i = \min_{j \in \Omega, i \neq j} \|\mathbf{x}_i - \mathbf{x}_j\| \tag{34}$$

With the separation distance defined, the algorithm for ordering refinement nodes is presented as follows:

1. Select the nodes in the base set
2. Add the base set to the active set: the set of boundary points which have already been selected
3. Evaluate the separation distance of the remaining boundary nodes to the nodes in the active set
4. Add the point with the largest separation distance to the active set
5. Repeat steps 3 and 4 until all boundary points have been added to the active set

Figure 16 shows an example of this process on a 2D mesh from Kedward et al. [2]. Assuming the full mesh in Figure 16(a) is equally spaced, the separation distances for the middle top, left, right and bottom points would all be equal, and they could be added to the active set in an arbitrary order.



(a) Full mesh (b) Coarse subset (c) First refinement (d) Second refinement

Figure 16: Example of the refinement node ordering algorithm from Kedward et al. [2]

When the boundary points are added to the active set, they are assigned a support radius equal to their current support distance. This means that the influence of the refinement nodes does not extend to any of the base nodes. Additionally, a refinement node at one point in the list will not influence any refinement node

ordered before it.

Given that the sorting of the boundary points is now defined, along with the assignment of the support radii, the resulting system of equations can now be examined. The equivalent of $\mathbf{\Phi}$ can be expressed as:

$$\mathbf{\Phi} = \left(\begin{array}{cccc|cccc} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,N_b} & 0 & 0 & \cdots & 0 \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,N_b} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_{N_b,1} & \phi_{N_b,2} & \cdots & \phi_{N_b,N_b} & 0 & 0 & \cdots & 0 \\ \hline \phi_{N_b+1,1} & \phi_{N_b+1,2} & \cdots & \phi_{N_b+1,N_b} & \phi_{N_b+1,N_b+1} & 0 & \cdots & 0 \\ \phi_{N_b+2,1} & \phi_{N_b+2,2} & \cdots & \phi_{N_b+2,N_b} & \phi_{N_b+2,N_b+1} & \phi_{N_b+2,N_b+2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_{N_s,1} & \phi_{N_s,2} & \cdots & \phi_{N_s,N_b} & \phi_{N_s,N_b+1} & \phi_{N_s,N_b+2} & \cdots & \phi_{N_s,N_s} \end{array}\right) \tag{35}$$

where N_b is the number of nodes in the base set, and N_s is the number of boundary (surface) points. The upper left submatrix is the same matrix that would arise from taking only the subset of N_b base nodes from the boundary, and using the basic RBF methodology. The lower left submatrix contains the influences of each of the base set nodes on the refinement nodes. Correspondingly, the upper right matrix is the influence of the refinement nodes on the base set which, due to the choice of their support radii, is a zero matrix. The lower right matrix is the influence of the refinement nodes on each other. Considering that the support radius of each refinement node is defined as the separation distance to the active set, each refinement node cannot influence any of the active set points when it is selected. Hence, the upper right triangle of this submatrix is zero. The equivalent to Equation (65) can be expressed as:

$$\begin{pmatrix} \mathbf{B} \\ \mathbf{R} \end{pmatrix} = \begin{pmatrix} \mathbf{\Phi}_b & \mathbf{0} \\ \mathbf{\Phi}_r & \mathbf{L} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} \tag{36}$$

Where $\mathbf{B}$ and $\mathbf{R}$ are the base node and refinement node coordinates respectively. $\mathbf{\Phi}_b$, $\mathbf{\Phi}_r$ and $\mathbf{L}$ are the three nonzero submatrices discussed above. The sub-system $\mathbf{B} = \mathbf{\Phi}_b\boldsymbol{\alpha}$ may be solved through inversion of $\mathbf{\Phi}_b$ as per the standard RBF methodology presented in Section 4.1. This leaves the bottom row of the Equation (36) to be solved for $\boldsymbol{\beta}$:

$$\mathbf{R} = \begin{pmatrix} \mathbf{\Phi}_r & \mathbf{L} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} \tag{37}$$

This can be done quickly thanks to the lower triangular structure of $\mathbf{L}$ [2]. The "residual" for the refinement points, $\mathbf{\Delta}_r = \mathbf{R} - \mathbf{\Phi}_r\boldsymbol{\alpha}$, can be evaluated first, which allows Equation (37) to be expressed as:

$$\mathbf{L}\boldsymbol{\beta} = \mathbf{\Delta}_r \tag{38}$$

Examining the expanded form of Equation (38) gives a clear path to evaluate $\boldsymbol{\beta}$.

$$\begin{pmatrix} \phi_{N_b+1,N_b+1} & 0 & \dots & 0 \\ \phi_{N_b+2,N_b+1} & \phi_{N_b+2,N_b+2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N_b+2,N_b+1} & \phi_{N_b+2,N_b+2} & \dots & \phi_{N_s,N_s} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{N_r} \end{pmatrix} = \begin{pmatrix} \Delta_{r_1} \\ \Delta_{r_2} \\ \vdots \\ \Delta_{r_{N_r}} \end{pmatrix} \tag{39}$$

Starting with the first row of Equation (39):

$$\phi_{N_b+1,N_b+1} \cdot \beta_1 = \Delta_{r_1} \tag{40}$$

β_1 can be evaluated as all other variables are known. Row two takes the form:

$$\phi_{N_b+2,N_b+1} \cdot \beta_1 + \phi_{N_b+2,N_b+2} \cdot \beta_2 = \Delta_{r_2} \tag{41}$$

which gives β_2 now that β_1 is known. The rest of the $\boldsymbol{\beta}$ coefficient matrix can then be evaluated in this manner. With the values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ known, an expression for the displacement or location of the nodes in the volume of the mesh should be found. Analogously to Equation (63), this is done by evaluating the RBF influences of the boundary nodes at each of the volume nodes:

$$\mathbf{V} = \left(\begin{array}{cccc|cccc} \phi_{1,1} & \phi_{1,2} & \dots & \phi_{1,N_b} & \phi_{1,N_{b+1}} & \phi_{1,N_{b+2}} & \dots & \phi_{1,N_s} \\ \phi_{2,1} & \phi_{2,2} & \dots & \phi_{2,N_b} & \phi_{2,N_{b+1}} & \phi_{2,N_{b+2}} & \dots & \phi_{2,N_s} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_{N_v,1} & \phi_{N_v,2} & \dots & \phi_{N_v,N_b} & \phi_{N_v,N_{b+1}} & \phi_{N_v,N_{b+2}} & \dots & \phi_{N_v,N_s} \end{array}\right) \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{N_b} \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{N_r} \end{pmatrix} \tag{42}$$

or, in matrix form:

$$\mathbf{V} = \left(\begin{array}{c|c} \boldsymbol{\Psi}_{b,v} & \boldsymbol{\Psi}_{r,v} \end{array} \right) \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} \tag{43}$$

Considering that there will likely be millions of volume nodes in modern aerodynamics meshes, the memory required to store the boundary to volume influence matrix will be large, and simply calculating this matrix and evaluating Equation (42) would be slow and memory intensive. Instead, Kedward et al. suggest operating on the volume points differently according to the sets which exert influence on them [2]. This begins with preprocessing the volume nodes into three groups. With d_i being the wall distance of volume node i to the moving boundary, r_0 being the support radius of the base set, and r_1 being the support radius of the first refinement node, these groups are:

1. $d_i > r_0$ - Category 1 - Volume points that are beyond the influence of the base set, and hence are not influenced by the boundary motion

2. $r_0 \geq d_i > r_1$ - Category 2 - Volume points which may be within the influence of the base set, but are definitely beyond the influence of the refinement nodes.

3. $d_i \leq r_1$ - Category 3 - Volume points which may be influenced by the displacement of both the base set and the refinement set.

In considering the treatment of the various groups, it is helpful to consider the distribution of these points in the mesh. Figure 17 has been generated to illustrate this in the simple case of simulating flow around a 2D aerofoil section. The bottom row shows a view focusing on the mesh close to the aerofoil boundary, whereas the top row is further zoomed out to view some of the freestream mesh. The right column of sub-figures shows the number of boundary nodes exerting an influence on each volume point, and the left column shows which points in the mesh belong to categories 1, 2 and 3.

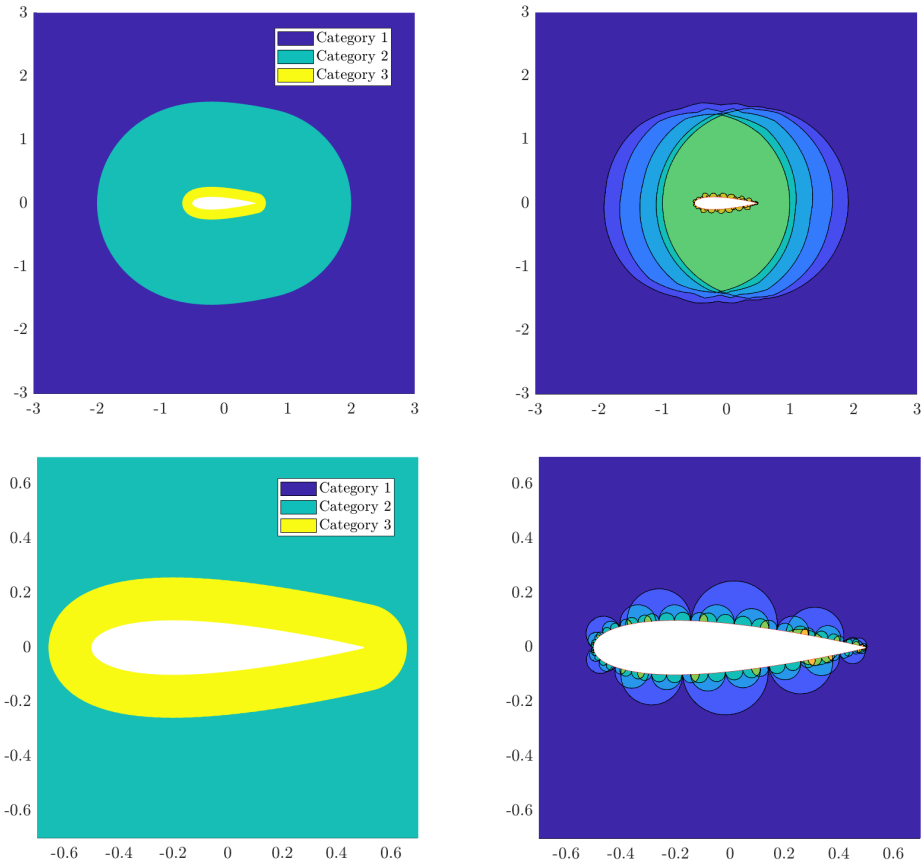


Figure 17: Comparison of the categorisation of volume mesh nodes and the number of influencing nodes (left column showing categorisation and right side the number of influences)

Considering Figure 17, we can see that, for applications with a freestream mesh, a notable proportion of the nodes will be category 1 volume points. As an example,

in the forward flight simulations in this thesis, approximately 33% of the nodes were category 1 nodes. This means that these points will not move and that no processing is required for these points, and hence they can be removed from Equation (42). The properties of category 2 points can also be used to simplify the calculations. Considering that these points are outside the influence of the refinement set, the values of the corresponding rows of $\boldsymbol{\Psi}_{r,v}$ are zero. Therefore, the evaluation of these rows can be eliminated, as well as the multiplication by $\boldsymbol{\beta}$, again saving computational time and memory. 34% of the category 2 points in the forward flight simulations were not acted upon by the base set. Lastly, it can be seen in Figure 17 that the mesh volume corresponding to the category 3 nodes incorporates the boundary layer mesh. This means that a significant proportion of the nodes will likely fall into this category. However, examining Figure 17 shows that many of these points are influenced by just a small number of refinement nodes, or sometimes none. This ultimately means that $\boldsymbol{\Psi}_{r,v}$ will be a very sparse matrix. For example, the simulations performed in this thesis have a $\boldsymbol{\Psi}_{r,v}$ with a sparsity of 99.87%. This means that a sparse matrix storage strategy can be used for $\boldsymbol{\Psi}_{r,v}$, further reducing memory requirements.

With the volume node categorisation in mind, the mathematical expressions for the volume node positions can be re-expressed as to clearly display the influences on each category of points. Where the subscripts 1, 2 and 3 refer to the volume node category, equivalents to Equation (43) are:

$$\mathbf{V}_2 = \boldsymbol{\Psi}_{b,v_2}\boldsymbol{\alpha} \tag{44}$$

$$\mathbf{V}_3 = \left(\begin{array}{c|c} \boldsymbol{\Psi}_{b,v_3} & \boldsymbol{\Psi}_{r,v_3} \end{array} \right) \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \boldsymbol{\Psi}_{b,v_3}\boldsymbol{\alpha} + \boldsymbol{\Psi}_{r,v_3}\boldsymbol{\beta} \tag{45}$$

The second expression for $\mathbf{V}_3$ is more specific to a code implementation due to the use of sparse matrix storage methods for $\boldsymbol{\Psi}_{r,v_3}$, meaning the two $\boldsymbol{\Psi}_{v_3}$ matrices should not be stored as one object. As the category 1 volume nodes are outside the support radius of any base or refinement point, there is no position change for these points (and hence no need for an equation for their position).

Kedward et al. provide a comparison between the operations required for the multiscale method and an RBF approach using the greedy point selection algorithm with a single correction update as detailed in Equation (27). They separate the computation aspects into four stages: the surface mesh preprocessing, volume mesh preprocessing, solve and update steps. They define these steps as follows [2]:

1. **Surface Preprocessing:** Greedy - Selecting the reduced set of surface points. Multiscale - Ordering the refinement points in order of separation distance.

2. **Volume Preprocessing:** Greedy - Selecting the nearest surface point for the error correction step. Multiscale - categorising the volume points according to which sets influence their displacement, as described above.

3. **Solve:** Greedy - Solving the full system with N_{red} base nodes. Multiscale - Solving the corresponding systems of equations for the base set and the refinement set coefficients $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

4. **Update:** Greedy - Moving the volume points and applying the correction factor. Multiscale - Moving the volume nodes according to the movement of the base and refinement sets.

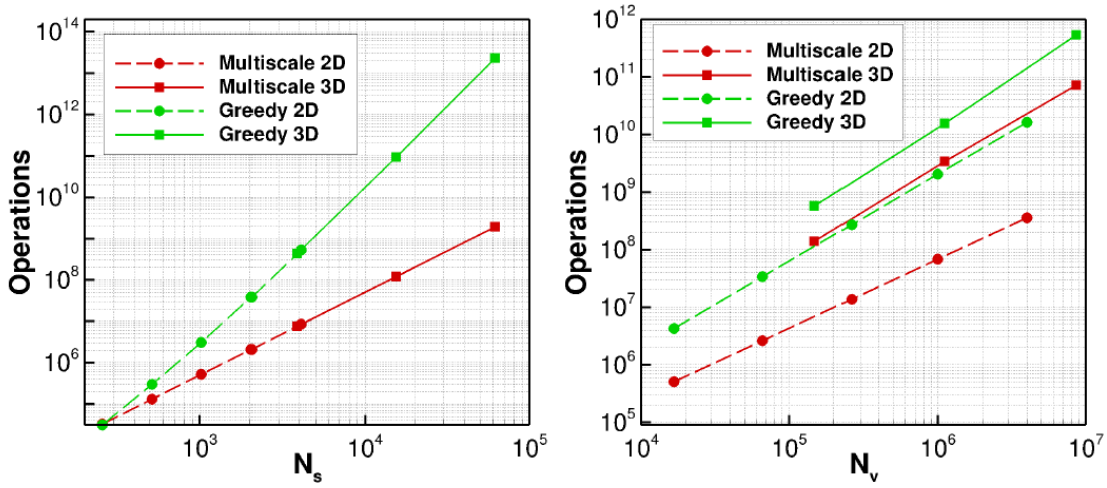
Table 3 from Kedward et al. compares the number of operations for the above steps [2]. N_s is the total number of nodes on the moving surface, N_b the number of base nodes, N_r the number of refinement nodes, and N_v the number of volume nodes.

Table 3: Comparison of the number of operations required for the greedy and multiscale RBF implementations [2]

Step	**Greedy**	**Multiscale**
Surface Preprocessing	$O(N_{\text{red}}^4)$	$O(N_s^2)$
Volume Preprocessing ($\alpha < 1$)	$O(\alpha N_v N_s)$	$O(\alpha N_v N_r)$
Solve	$O(N_{\text{red}}^3)$	$O(N_b^3)$
Update	$O(N_v N_{red})$	$O(N_v N_b)$

In most implementations it should be considered that N_b and N_{red} will be close in size. Both methods are trying to reduce computing cost by minimising the number of base nodes, and as these nodes are the most influential in determining mesh deformation, the number of base nodes required will be similar between methodologies. Resultingly, it can be seen that the solve and update steps are very similar in computing cost between the greedy and multiscale methods. However, the preprocessing costs shows the computational advantage of the multiscale method. The multiscale method has a small scaling advantage at the volume preprocessing stage as N_s must be larger than N_r. The comparison of surface preprocessing cost is hard to evaluate due to the comparison of N_{red} to N_s, but due to the differing exponents it is clear that the multiscale approach scales better for large systems. Kedward et al. display the advantage of the multiscale method in comparisons of the preprocessing stage for a number of 2D and 3D meshes. The 2D meshes ranged from 257 surface points by 65 points perpendicular to the surface to 4097×1025. The 3D meshes ranged from surface and volume node numbers of $N_s = 3881, N_v = 1.48 \cdot 10^5$ to $N_s = 61\ 601, N_v = 8.62 \cdot 10^6$. $N_b = N_{red} = 0.05 N_s$ for all meshes. The results of this comparison can be seen in Figure 18. While the greedy method had a small advantage over the multiscale method in surface mesh preprocessing on the smallest 2D mesh, the multiscale method was faster in every other comparison. Furthermore, the smallest mesh size is far too small for any serious aerodynamic study, and for the realistic mesh sizes the multiscale method is significantly faster.

Kedward et al. also performed a comparison between these two methods in terms of deformed grid quality. This used the Brite-Euram multidisciplinary optimisation



(a) Surface mesh preprocessing

(b) Volume mesh preprocessing

Figure 18: Comparison of preprocessing costs between a greedy algorithm and the multiscale method with $N_b = N_{\text{red}}$ from Kedward et al. [2]

wing as the surface geometry, and the RBF implementations used Wendland's C2 function with a support radius of one semi-span [2]. The error correction step of the greedy implementation was based on a single point and employed a support radius equal to the separation distance of the first refinement node, in order to enable a comparable setup. The average orthogonality of the total mesh and mesh near the wing surface were calculated for different $N_{\text{red}} = N_b$ with four different sized volume meshes. The results for the largest mesh, with $N_v = 8.620$ million, can be seen in Figure 19. The results for the other mesh sizes followed a similar trend. It can be seen that the multiscale algorithm outperforms the greedy algorithm across all N_b, N_{red}. At $N_b \approx 100$ the average orthogonality is 0.23% greater across the whole mesh, and 1.4% greater in the near-surface mesh. At $N_b \approx 500$ this advantage is reduced by 94% for the mesh-average orthogonality, and by 90% for the near-surface mesh. Both methods are practically equal in deformation quality above $N_b = 1000$. Also of note is that both methods experienced diminished returns above $N_b = 500$.

3.9 Iterative Methods for Greedy Radial Basis Function Algorithms

As discussed in the previous two sections, greedy algorithms produce close to optimal deformation quality when using only a given number of base set nodes. However, this comes at the cost of potentially very computationally expensive preprocessing of order $O(N_{\text{red}}^4)$ [2]. Whilst there have been efforts to reduce the cost per iteration of greedy algorithms, the computation saving measures tend to lead to a different, less-optimal point selection than the standard greedy algorithm. However, there are methods that decrease the computational loads without affecting the point selection. This begins with noticing that greedy algorithms add points to the reduced base set each iteration, and thus most of these nodes were already present on the previous

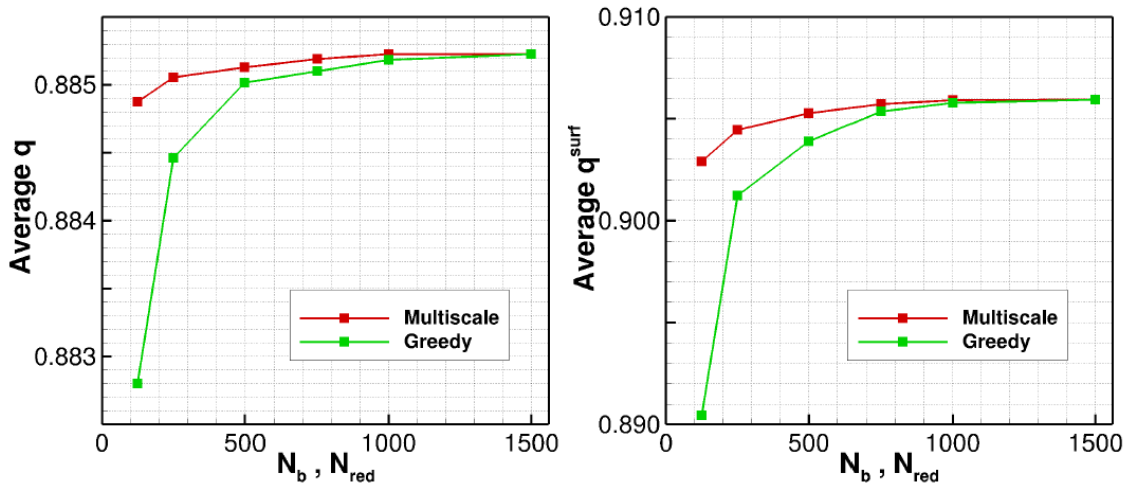


Figure 19: Variation in average mesh orthogonality between multiscale and greedy RBF methods (left - total mesh, right - near surface mesh) [2]

iteration. As a result, researchers developed iterative methods that append data to many of the RBF mesh motion objects, and also perform computations cheaper than recalculating each object from scratch each iteration. The specific vectors or matrices subject to these iterative methods depend on the underlying RBF methods, however typical targets include the list of base set nodes, support radii, base set to base set influence matrices and their inverses.

A method utilising block iteration and parallelisation for RBF mesh deformation was demonstrated by Zhao et al. [3]. Using three test cases of an undulating fish, the ONERA M6 Wing and a super-cavitating hydrofoil, they demonstrated that the vast majority of computational time is spent on inverting $\boldsymbol{\Phi}$, creating $\boldsymbol{\Psi}$, and the multiplication of $\boldsymbol{\Psi}$ by $\boldsymbol{\alpha}$. For example, in a test case selecting approximately 620 nodes from a surface mesh of 5646 nodes, calculating $\boldsymbol{\Phi}^{-1}$ took 48% of the computational time spent on the greedy point selection, creating $\boldsymbol{\Psi}$ took 33%, and evaluating $\boldsymbol{\Psi\alpha}$ took 14%. Zhao et al. start with a standard greedy implementation as described in Section 3.7. The first step in constructing their iterative method is expressing the base set to base set matrix at iteration i, $\boldsymbol{\Phi}_i$, as:

$$\boldsymbol{\Phi}_i = \begin{bmatrix} \boldsymbol{\Phi}_{i-1} & \boldsymbol{\nu} \\ \boldsymbol{\nu}^T & \phi(0) \end{bmatrix} \tag{46}$$

where $\boldsymbol{\Phi}_{i-1}$ is the base set to base set influence matrix from the previous iteration, and $\boldsymbol{\nu}$ is the vector of RBF values evaluated between the previous base nodes and the new node: $\boldsymbol{\nu} = [\phi_{1,i}, \phi_{2,i}, \phi_{3,i} \dots \phi_{i-1,i}]^T$. The matrix will be symmetric as $\phi(i,j) = \phi(j,i)$. This iterative construction method allows the computational complexity of constructing $\boldsymbol{\Phi}_i$ to be reduced from $O(n^3)$ to $O(n^2)$ [3].

Constructing $\boldsymbol{\Phi}_i^{-1}$ iteratively was based on identities presented by Lu and Shiou in [88]. The identity:

$$\mathbf{\Phi}_i \mathbf{\Phi}_i^{-1} = \mathbf{I}_i \tag{47}$$

where $\mathbf{I}_i$ is a $i \times i$ identity matrix, can be expressed in block matrix form as:

$$\begin{bmatrix} \mathbf{\Phi}_{i-1} & \boldsymbol{\nu} \\ \boldsymbol{\nu}^T & \phi(0) \end{bmatrix} \begin{bmatrix} \boldsymbol{M}_{i-1} & \boldsymbol{\eta} \\ \boldsymbol{\eta}^T & p \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{i-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{48}$$

where $\mathbf{0}$ is a zero vector of length $i-1$. Solving this equation results in an iterative expression of the inverse:

$$\mathbf{\Phi}_i^{-1} = \begin{bmatrix} \mathbf{\Phi}_{i-1}^{-1} + c\mathcal{B}\mathcal{B}^T & c\mathcal{B} \\ c\mathcal{B}^T & c \end{bmatrix} \tag{49}$$

with the definitions of c and $\mathcal{B}$:

$$\mathcal{B} = -\mathbf{\Phi}_{i-1}^{-1} \boldsymbol{\nu} \tag{50}$$

$$c = \frac{1}{\phi(0) - \boldsymbol{\nu}^T \mathbf{\Phi}_{i-1}^{-1} \boldsymbol{\nu}} \tag{51}$$

This technique is the same as the iterative method proposed by Skala in his previously published work [89]. This algorithm reduces the computational complexity of this step from $O(n^3)$ to $O(n^2)$. Zhao et al. also note a special case, that when using RBFs where $\phi(0) = 0$, the initial $\mathbf{\Phi}_1$ evaluates to 0, and hence has no inverse [3]. In this case, $\mathbf{\Phi}_2$ should be constructed and inverted non-iteratively.

Zhao et al. also present an iterative method for the construction of $\mathbf{\Psi}$. Starting with observing the structure of the matrix, it can be seen that it consists of rows evaluating all base set influences at a specific surface point, and columns evaluating the influence of a specific base set node on all the surface nodes:

$$\mathbf{\Psi} = \begin{bmatrix} \phi(\|\mathbf{x}_{b_1} - \mathbf{x}_{c_1}\|) & \phi(\|\mathbf{x}_{b_1} - \mathbf{x}_{c_2}\|) & \dots & \phi(\|\mathbf{x}_{b_1} - \mathbf{x}_{c_{N_c}}\|) \\ \phi(\|\mathbf{x}_{b_2} - \mathbf{x}_{c_1}\|) & \phi(\|\mathbf{x}_{b_2} - \mathbf{x}_{c_2}\|) & \dots & \phi(\|\mathbf{x}_{b_2} - \mathbf{x}_{c_{N_c}}\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|\mathbf{x}_{b_{N_b}} - \mathbf{x}_{c_1}\|) & \phi(\|\mathbf{x}_{b_{N_b}} - \mathbf{x}_{c_2}\|) & \dots & \phi(\|\mathbf{x}_{b_{N_b}} - \mathbf{x}_{c_{N_c}}\|) \end{bmatrix} \tag{52}$$

Since the set of boundary points is constant, but the set of base nodes grows by one each iteration of the greedy algorithm, the growth of $\mathbf{\Psi}$ can be seen as appending a new column on the right of this matrix each iteration. Therefore $\mathbf{\Psi}_i$, the matrix $\mathbf{\Psi}$ at step i, can be expressed in terms of $\mathbf{\Psi}_{i-1}$:

$$\mathbf{\Psi}_i = \left[\mathbf{\Psi}_{i-1}, \boldsymbol{v}\right] \tag{53}$$

where $\boldsymbol{v}$ is the vector $\boldsymbol{v} = [\phi(\|\mathbf{x}_{b_1} - \mathbf{x}_{c_i}\|), \phi(\|\mathbf{x}_{b_2} - \mathbf{x}_{c_i}\| \dots \phi(\|\mathbf{x}_{b_{N_b}} - \mathbf{x}_{c_i}\|)]^T$. This again reduces the computational complexity from $O(n^3)$ to $O(n^2)$ [3].

Beyond the speedup gained from the iterative methods, Zhao et al. also derive increased execution speed from the parallelisation of the calculation of the boundary

point positions at each iteration. Equation (54) shows the mathematics governing this movement.

$$\Delta \mathbf{V} = \boldsymbol{\Psi}\boldsymbol{\alpha}$$
$$= \begin{bmatrix} \phi(\|\mathbf{x}_{v_1} - \mathbf{x}_{b_1}\|) & \phi(\|\mathbf{x}_{v_1} - \mathbf{x}_{b_2}\|) & \dots & \phi(\|\mathbf{x}_{v_1} - \mathbf{x}_{b_{N_b}}\|) \\ \phi(\|\mathbf{x}_{v_2} - \mathbf{x}_{b_1}\|) & \phi(\|\mathbf{x}_{v_2} - \mathbf{x}_{b_2}\|) & \dots & \phi(\|\mathbf{x}_{v_2} - \mathbf{x}_{b_{N_b}}\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|\mathbf{x}_{v_{N_v}} - \mathbf{x}_{b_1}\|) & \phi(\|\mathbf{x}_{v_{N_v}} - \mathbf{x}_{b_2}\|) & \dots & \phi(\|\mathbf{x}_{v_{N_v}} - \mathbf{x}_{b_{N_b}}\|) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{N_b} \end{bmatrix} \tag{54}$$

The location of each boundary node is dependant only on the original node coordinates $\mathbf{x}_{v_i}$, base set node coordinates $\mathbf{x}_b$, and the coefficient matrix $\boldsymbol{\alpha}$. Therefore, each boundary node can be distributed to different processors, along with the appropriate rows of $\boldsymbol{\Psi}$ and a copy of $\boldsymbol{\alpha}$.

Skala also presents a method for iteratively calculating $\boldsymbol{\Phi}_i$ when removing a point from $\mathbf{x}_{\text{red}}$. It is conceivable that a greedy-style algorithm may remove points from the reduced base set, in an effort to produce a more optimal selection for a given number of reduced base set nodes (as previously stated, greedy algorithms optimise the node selection specifically at each iteration, and therefore the selections are not guaranteed to be the same as the optimal choice of n nodes selected at once). However, to the author's knowledge there are no such algorithms.

3.10 Comparison of Radial Basis Function Methods to other Arbitrary Lagrangian-Eulerian Methods

Numerous ALE methods have been developed with different target applications and to prioritise various aspects of simulations, for example computational efficiency, grid quality or robustness. However, as there is a wide variety of applications for most methods, comparing ALE methods against each other is beneficial for evaluating which are best suited to certain uses. As the primary concerns with mesh deformation methods are typically the computational feasibility of the simulation and its accuracy, the comparisons usually measure the time taken to deform a mesh and the quality of the deformed mesh.

Estruch et al. compared the RBF method and the spring analogy in simulations of a duct with a moving indentation [78]. The simulation was performed on an unstructured mesh with 72 300 control volumes. Mesh quality was analysed through the measure of aspect ratio, with the results presented in Figure 20. Whilst the graph shows the sensitivity of the RBF method to the support radius, both support radii maintain a higher aspect ratio than the spring analogy from iteration 22 onward. Furthermore, when considering the simulations with a support radius of 1, the spring analogy has only a marginal advantage in the early iterations, and significantly worse performance in the later iterations.

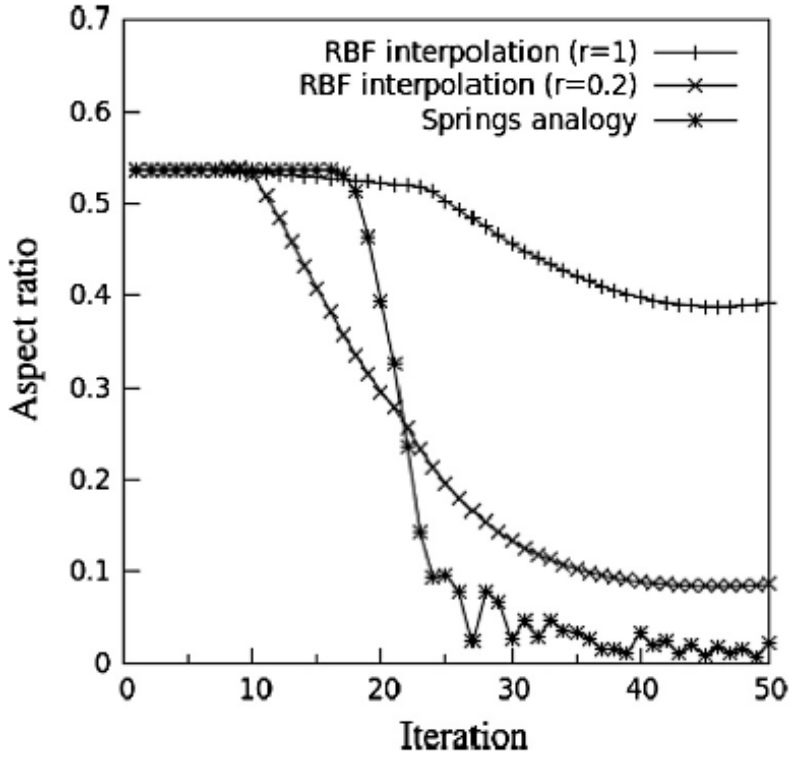


Figure 20: Comparison of RBF and spring analogy methods with regards to aspect ratio from Estruch et al. [78]

Table 4: Comparison of computational time required for one iteration in the simulation of a duct with a moving indentation from Estruch et al. [78]

CPUs	**RBF (s)**	**Spring Analogy (s)**	**RBF to Spring Analogy Fraction**
1	21.8	2780	0.00783
8	2.72	429	0.00635
16	1.36	233	0.00584

Estruch et al. also compare the computing time for one iteration of the mesh motion between the two techniques for computations with 1, 8 and 16 CPUs. This data, and the fraction of the Spring Analogy deformation time required by the RBF method, are presented in Table 4. This data shows a significant computational advantage for the RBF method both in terms of absolute time taken, and in superior scaling with increasing CPUs.

Bos et al. compare RBF mesh motion to Laplacian PDE mesh motion and a solid body rotation stress (SBR stress) equation technique [59]. A number of RBF methods were compared, with the "absolute implementation" being the full surface base node method as described in Section 3.5; the "relative implementation" being the same method, but recalculating the RBF system each iteration from the new, deformed mesh; and the third method being the relative implementation with

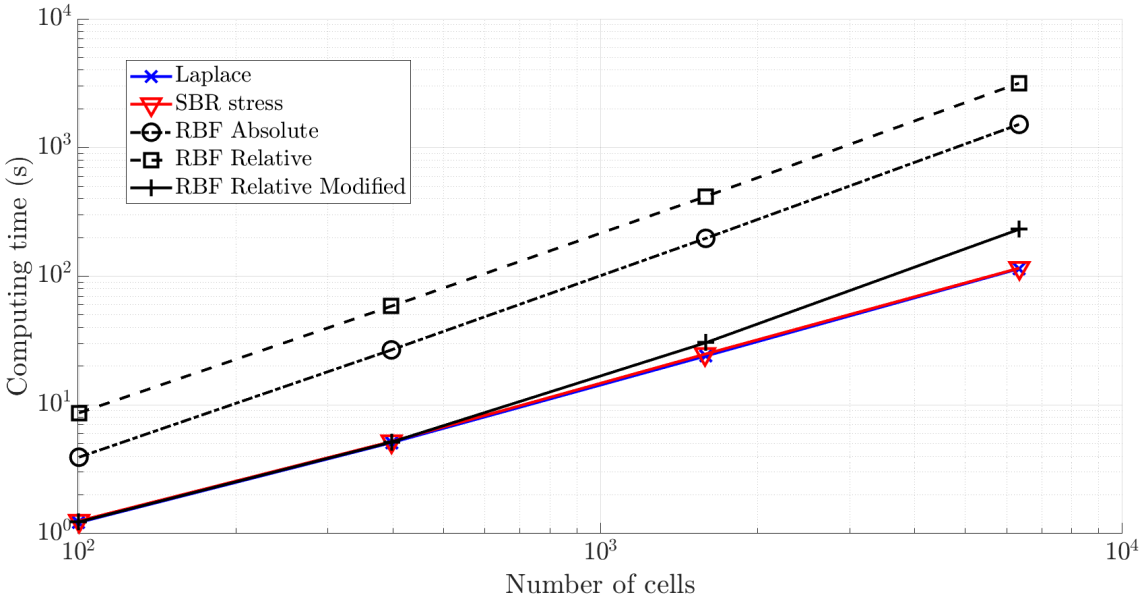


Figure 21: Comparison of RBF, Laplace and solid body rotation stress mesh motion methods with regards to computing time for deforming a mesh around a rotating block. Data and graph layout from Bos et al. [59].

additional procedures to use a reduced base node set. These methods were tested by deforming a two-dimensional grid containing a rotating block. Figure 21 shows the results of these tests. There is a reduction in computational time from 86% at approximately 100 cells to 93% at approximately 6 300 cells when integrating the reduced base set methods. Considering the improvement to the relative RBF method, this would likely make the more common absolute method competitive with Laplace PDE mesh motion.

With regards to mesh quality, Bos et al. compare the Laplace and SBR stress mesh motion techniques to the RBF method using multiple RBFs, with Wendland's C2 and thin plate spline functions being of particular interest. Mesh quality was evaluated with the maximum and average values of cell skewness and nonorthogonality. Lower values of these measures are better, with possible skewness values between 0 and 1, and possible nonorthogonality values between 0° and 90°. The methods were tested with the case of a rotating and translating block with fixed mesh boundaries. The skewness and nonorthogonality results from the tests are presented in Tables 5 and 6 respectively. The RBF methods show a notable advantage over the other methods when comparing the skewness values, as well as better maximum nonorthogonality values. The thin plate spline RBF method also performed better regarding average nonorthogonality, but the simulation with Wedland's C2 function was 16% worse. Beyond the maximum and average values, the distribution of cell quality is important. Figure 22 shows the nonorthogonality values for the rotation and displacement test case with Laplace, SBR and RBF mesh motion. The thin plate spline RBF was used for the RBF method. This visualisation shows a clear superiority of the RBF value to maintain mesh quality next to the moving surface, particularly in comparison to the Laplace mesh motion, where some cell internal

Table 5: Skewness values and differences from Bos et al. [59]. Differences are relative to Laplace mesh motion.

Method	**Maximum**	**Difference(%)**	**Average**	**Difference (%)**
Laplace	0.95	0	0.09	0
SBR Stress	0.96	0.7	0.08	-4
RBF C2	0.65	-32	0.065	-28
RBF TPS	0.52	-45	0.051	-41

Table 6: Nonorthogonality values and differences from Bos et al. [59]. Differences are relative to Laplace mesh motion.

Method	**Maximum**	**Difference (%)**	**Average**	**Difference (%)**
Laplace	72.1	0	20.1	0
SBR Stress	75.1	4	21.3	6
RBF C2	59.6	-17	23.4	16
RBF TPS	52.5	-27	19	-6

angles are close to 180°.

Wang et al. compare the RBF and Delaunay Graph methods to the Delaunay Graph-RBF methods they propose [66] (see Section 3.2.7 for details). These are evaluated with regards to computing time and mesh quality. Quality was assessed with a skew-size metric with value 1 if the size and shape of the element were unchanged, and a value of 0 for degenerate elements. The methods were first tested with the rotation and translation of a 2D block in a structured mesh. The average and minimum values of the size-skew metric are listed in Table 7. While the RBF method had the highest average mesh quality, the minimum quality was the lowest of the four methods. Wang et al. also note that the RBF method preserves mesh quality well near the boundary, but the quality of the deformation can degrade further from the boundary.

Wang et al. performed another test in which a sphere was squeezed within a surrounding structured grid. The average skew-size metric values were 0.837, 0.858 and 0.862 for the Delaunay Graph, Delaunay Graph RBF, and RBF methods respectively. The computing time for the mesh deformation with 201 920 total nodes and 10 096 boundary nodes was 0.113s for the Delaunay graph method, 0.161s for

Table 7: Average and minimum values of the skew-size metric in tests of a rotated and translated block in a structured mesh from Wang et al. [66]

Method	**Average**	**Minimum**
Delaunay RBF	0.821	0.177
Delaunay RBF Rotational	0.858	0.237
Delaunay	0.816	0.153
RBF	0.861	0.0838

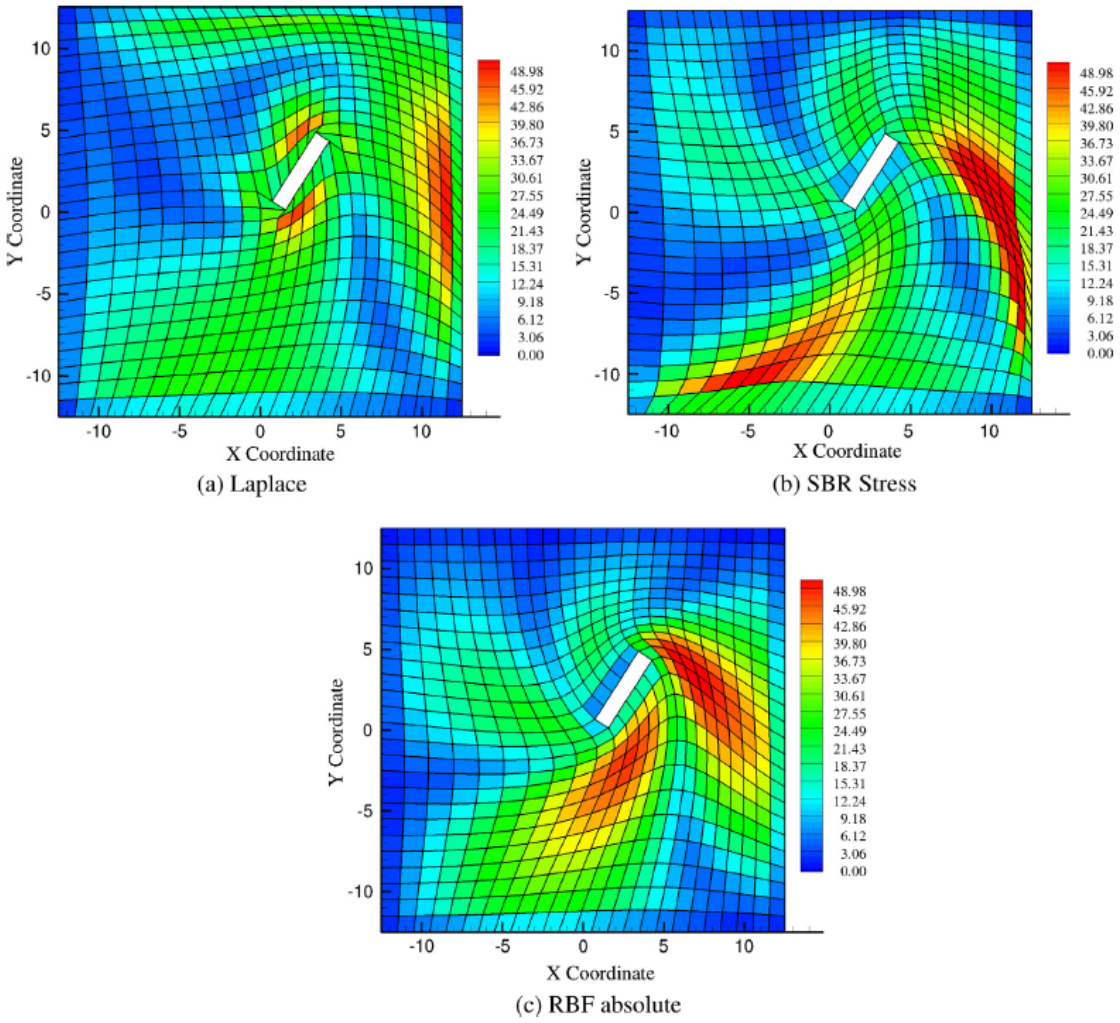


Figure 22: Comparison of nonorthogonality between RBF, Laplace and solid body rotation stress mesh motion methods from Bos et al. [59].

Table 8: Comparisons of CPU time to deform an Euler mesh from Jakobsson and Amoignon [82]

Measure	**Smoother**		**RBF**		
	tol = 10^{-14}	**tol = 10^{-7}**	**2166**	**1378**	**730**
Matrix inversion	0	0	34	9	3
Deformation	66	36	17	9	4
Total	66	36	51	18	7
Fraction	1	0.545	0.773	0.273	0.106

the Delaunay Graph RBF method, and 11 784s for the RBF method.

Jakobssen and Amoignon performed tests comparing RBF mesh motion to a Laplace smoother mesh motion technique that is similar to the spring analogy, in that each node has scaled connected edge length changes that sum to zero [82, 90]. The CPU time required to deform a mesh was tested using both of these methods with two different cases. The first of these was on a "Euler mesh" of approximately 136 000 nodes, suitable for inviscid simulations, and the second was on a "Navier-Stokes mesh" of approximately one million nodes, suitable for simulation of a viscid flow. The Laplace smoother solved for the new node locations using a diagonal preconditioned conjugate gradient method, with two different convergence tolerances used for the new node positions: 10^{-7} and 10^{-14}. The CPU timings for the tests with the two meshes are presented in Tables 8 and 9. The numbers under the RBF headers list the number of base nodes used. All RBF simulations used the inverse quadratic RBF. The fraction row shows the total CPU time of each method relative to the Laplace smoother with tolerance 10^{-14}. The RBF method was quicker than the Laplace smoother for all conditions, except the RBF with 2166 base nodes on the Euler mesh against the Laplace smoother with tolerance 10^{-7}. Furthermore, it can be seen that the deformation stage of the RBF methods was quicker than the Laplace smoothers. This is particularly noticeable for the RBF runs with the largest number of base nodes, where the deformation times were 33% and 42% of the total time on the Euler and Navier-Stokes meshes respectively. Additionally, these deformation steps took less time than the corresponding step of the Laplace smoother with tolerance 10^{-7}, being only 47% of the Laplace smoother's time on the Euler mesh, and 24% on the Navier-Stokes mesh. This suggests that these RBF methods may be superior to the Laplace smoother and similar methods when used in simulations with time-varying deformation. As the inverse quadratic RBF has global support, the deformation performance may be further improved, especially on large meshes, through using an RBF with compact support, which may not operate on all mesh nodes. Jakobsson and Amoignon also add that numerical solutions to methods employing elliptic equations, such as the Laplace smoother and many PDE mesh motion methods [68, 82], perform worse on highly stretched meshes, as typically seen with boundary-layer resolving meshes. In contrast, the RBF methods are agnostic to the mesh structure.

Selim and Koomullil surveyed the popular techniques for mesh motion in the

Table 9: Comparisons of CPU time to deform a Navier-Stokes mesh from Jakobsson and Amoignon [82]

Measure	**Smoother**		**RBF**		
	tol = 10^{-14}	**tol = 10^{-7}**	**2166**	**1378**	**730**
Matrix inversion	0	0	173	9	1
Deformation	1275	515	123	35	17
Total	1275	515	296	44	18
Fraction	1	0.404	0.232	0.0345	0.0141

Table 10: Computational complexity of various ALE methods as per Selim and Koomullil [91]

Method	**Complexity**
TFI	$O(N_{\text{vert}})$
Linear Springs	$O(N_e^3)$
Torsional Spring	$O(N_e^3 + N_{\text{vert}}^3)$
Laplace PDE	$O(N_{\text{vert}})$
Delaunay Graph	$O(N_d + N_{\text{vert}}^{1/3})$
RBF	$O(N_b^3)$
RBF with Greedy Algorithm	$O(N_{\text{red}}^3)$

two decades up to 2016 [91]. This included the linear and torsional spring analogies, Laplacian PDE mesh motion, transfinite interpolation, Delaunay Graph mesh motion, RBF methods, and RBF methods with greedy algorithm point base node selection. Within this survey, they list the computational complexity of these algorithms as in Table 10. In this table, N_e is the number of edges, N_{vert} is the number of vertices and N_d is the number of vertices in the Delaunay graph. Although the methods scale with different variables, TFI and Laplace PDE mesh motion both have linear asymptotic scaling, and the Delaunay Graph method only exceeds this by only $N_{vert}^{1/3}$. In contrast, the RBF methods scale with the cube of the number of base nodes, which will be correlated to the number of surface points. As discussed in Section 3.8, RBF methods scale linearly with the volume node number. This suggests that the size of the mesh can be increased without serious impact on the computing requirements, if the number of base nodes is kept constant. This could, for example, involve refining a mesh in the wake region. The computational complexities of the spring analogy methods suggest that increasing the number of volume nodes in this way would have a more significant computational impact. Selim and Koomullil present RBF methods with greedy base node selection as a balanced option and the only method in the survey having "no major disadvantage".

3.11 Summary

The issue of simulating moving or deforming objects is prominent within the area of computational fluid dynamics, given the commonality of such scenarios. There

are three prominent approaches to this issue: chimera meshes, immersed boundary methods and Arbitrary Lagrangian-Eulerian methods. Whilst chimera methods are very adaptable for rigid motions, the need of an ALE method for deforming bodies and the complexity of hole cutting and inter-grid interpolation methods somewhat limit their appeal. IBMs promise extremely high flexibility to simulation scenarios, but the significant computational costs have so far prevented widespread adoption of these methods, not to mention the additional requirements of AMR and dynamic load balancing methods, and difficulties in boundary layer treatment. ALE methods have seen broader adoption due to their relative simplicity and ease of integration. Of the ALE methods, RBF methods have been growing in popularity as a result of a number of factors. Compared to the spring analogy, mesh connectivity information is not required and an expensive calculation of grid node locations is not necessary each time the mesh is moved. Compared to Delaunay Mapping mesh motion, the deformed meshes present higher quality and more robustness to larger movements. RBF methods avoid the complexities of PDE mesh motion methods in solving the PDEs and in tuning these equations to produce the desired behaviour. Whilst algebraic mesh motion was popular, RBF methods show more flexibility in accommodating the increased complexity of modern meshes and the growing popularity of unstructured meshes.

Since its first use by de Boer et al. in 2007 [67], RBF mesh motion has seen numerous developments. As mesh sizes increased in order to target higher-fidelity aerodynamic simulation, it was realised that the use of all the surface mesh nodes of the moving body was impractical with regards to computational cost and numerical accuracy. Use of reduced base sets lead to the development of greedy methods, which provide a close to minimal point selection for a given measure of error over the surface mesh. Greedy methods are expensive when selecting large numbers of base nodes, which prompted the development of reduced-cost greedy methods. Whilst techniques such as selecting multiple points per iteration or using approximate solutions do affect the idealty of the solution, iterative methods successfully produce the same point choices as the standard greedy algorithms while reducing the asymptotic computational cost of many stages by one power. Kedward et al. presented the multiscale method in 2017 [2], which provided a way to specify the complete surface location, whilst avoiding the numerical issues of using all the nodes as base nodes and maintaining a reasonable computational cost.

When looking at comparisons of RBF methods to other ALE methods, the literature presents a favourable view of RBFs in relation to mesh quality, while showing that they are nonetheless competitive with regards to computational costs. In particular, the RBF methods were shown to generally produce higher quality meshes than the popular spring analogy and Laplace PDE mesh motion, especially with note to the near-boundary mesh. Whilst timings of these methods were often limited to relatively small meshes by current standards, RBF methods outperformed the spring analogy and the similarly structured Laplace smoother method. The performance was similar to Laplace PDE mesh motion. Whilst Delaunay Graph mesh motion was much quicker than RBF mesh motion, it is often not considered a

viable alternative due to the mesh quality limitations when performing large deformations. There is also some limited evidence that RBF methods may be superior when considering increases in wake mesh refinement, highly stretched meshes and time-varying deformations.

4 Radial Basis Function Methodologies

This chapter covers the mathematical and technical details of the radial basis function techniques used in this thesis. The chapter begins with a thorough mathematical description of RBF mesh motion, before covering the variable base node support radii technique that was used to improve the mesh deformation of helicopter rotors. A novel technique of an iterative, greedy, multiscale RBF approach is then introduced, detailing the iterative methods and new approach to the greedy error criterion. An asymptotic cost analysis is performed, displaying the computational savings of the iterative methods over a naive combination of greedy and multiscale methods.

4.1 Radial Basis Function Mesh Motion

The RBF mesh motion methodology can be seen as the mapping of the displacement, defined by the movement of the base set, to the rest of the simulation domain. Firstly, it is assumed that this problem can be broken down with a dimension-by-dimension approach. Correspondingly, the mapping function is defined as a vector of functions, each of which correspond to one of the spatial dimensions:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} F_x(\mathbf{x}) \\ F_y(\mathbf{x}) \\ F_z(\mathbf{x}) \end{bmatrix} \tag{55}$$

This mapping function can be more intuitively thought of as a function that describes the sum of displacement influences from each base node at an arbitrary point in space. Each of the functions $F_i(\mathbf{x})$ is defined to be a sum of radial basis functions over the number of base nodes N_b, as per Equation (56):

$$F_i(\mathbf{x}) = \sum_{j=1}^{N_b} \alpha_j^i \phi(\|\mathbf{x} - \mathbf{x}_j\|) \tag{56}$$

where α_j^i is a scalar corresponding to dimension i and the jth structural node. $\|\cdot\|$ is a chosen norm on $\mathbb{R}^3$, and ϕ is a radial basis function. The RBF used in the simulations in this thesis is Wendland's C2 function. It was chosen due to it being successfully used in a wide range of publications [72, 59, 2, 79, 76, 77, 68, 67, 63]; it being a low-order, compactly supported RBF with a zero derivative at $r = 0$ (the benefits of these properties are described in Section 3.4); as well as testing on the scenarios presented in this thesis. A polynomial p^i may be added to Equation (56) as well, giving:

$$F_i(\mathbf{x}) = \sum_{j=1}^{N_s} \alpha_j^i \phi(\|\mathbf{x} - \mathbf{x}_j\|) + p^i(\mathbf{x}) \tag{57}$$

Whilst the polynomial may be used to guarantee uniqueness of the interpolation and to recover rigid body movements [92], it was not used in the implementations in this thesis. As Wendland's C2 function is positive definite, the polynomial is not needed for matrix conditioning, and testing showed no increase in the quality of the

deformed mesh when using the polynomial.

Next, the observation is made that the desired mapping should return the locations of the base set when their locations are given as input into the function. With $\mathbf{b}$ being the coordinates of a base set node, this can be expressed as:

$$\mathbf{F}(\mathbf{b}) = \mathbf{b} \tag{58}$$

and is true for all $\mathbf{b}$. Taking the definition of F_i in Equation (56), the expressions can be evaluated at each base set node and then "stacked" to form the system in Equation (59).

$$\begin{pmatrix} b_1^i \\ \vdots \\ b_{N_b}^i \end{pmatrix} = \begin{pmatrix} \phi_{1,1} & \dots & \phi_{1,N_b} \\ \vdots & \ddots & \vdots \\ \phi_{N_b,1} & \dots & \phi_{N_b,N_b} \end{pmatrix} \begin{pmatrix} \alpha_1^i \\ \vdots \\ \alpha_{N_b}^i \end{pmatrix} \tag{59}$$

where $\phi_{m,n} = \phi(\|\mathbf{x}_m - \mathbf{x}_n\|)$ and b^i is the coordinate of $\mathbf{b}$ in dimension i. This may be rewritten as:

$$\mathbf{B}^i = \mathbf{\Phi}\boldsymbol{\alpha}^i \tag{60}$$

Considering that the positions of the structural nodes are known, and therefore also $\phi_{m,n}$, the coefficient vector $\boldsymbol{\alpha}^i$ may be found using:

$$\boldsymbol{\alpha}^i = \mathbf{\Phi}^{-1}\mathbf{B}^i \tag{61}$$

As previously stated, the movement of the base set is used to map the displacement of the other nodes. As such, the same mapping function $F(\mathbf{x})$ is used to evaluate the locations of the volume mesh nodes in the deformed mesh:

$$F_i(\mathbf{v}_k) = \alpha_1^i \phi(\|\mathbf{v}_k - \mathbf{b}_1\|) + \dots + \alpha_{N_b}^i \phi(\|\mathbf{v}_k - \mathbf{b}_{N_b}\|) \tag{62}$$

Similarly to Equation (59), the evaluations of Equation (62) at each volume node can be collated:

$$\begin{pmatrix} v_1^i \\ \vdots \\ v_{N_v}^i \end{pmatrix} = \begin{pmatrix} \phi_{1,1} & \dots & \phi_{1,N_b} \\ \vdots & \ddots & \vdots \\ \phi_{N_v,1} & \dots & \phi_{N_v,N_b} \end{pmatrix} \begin{pmatrix} \alpha_1^i \\ \vdots \\ \alpha_{N_b}^i \end{pmatrix} \tag{63}$$

where $\phi_{m,n} = \phi(\|\mathbf{v}_m - \mathbf{b}_n\|)$. Analogously to Equation (60), this can be expressed in the form:

$$\mathbf{V}^i = \mathbf{\Psi}\boldsymbol{\alpha}^i \tag{64}$$

Using the expression for the coefficient vector $\boldsymbol{\alpha}^i$ from Equation (61) gives a way to evaluate $\mathbf{V}^i$, as it is only dependant on the coordinates of the base node set and the volume node set.

$$\mathbf{V}^i = \mathbf{\Psi}\boldsymbol{\alpha}^i = \mathbf{\Psi}\mathbf{\Phi}^{-1}\mathbf{B}^i = \mathbf{H}\mathbf{B}^i \tag{65}$$

Lastly, for reasons explained in Section 3.4, the RBFs typically used are zero outside the chosen support radius. This means that Equation (65) must be expressed in terms of the change in positions, instead of absolute positions:

$$\Delta \mathbf{V}^i = \mathbf{H} \Delta \mathbf{B}^i \tag{66}$$

To understand why, consider a volume point beyond the support radii of all structural nodes. As $\phi(\|\mathbf{v}_k - \mathbf{b}_n\|)$ would evaluate to zero for all $\mathbf{b}_n$, so would the corresponding row of $\boldsymbol{\Psi}$, and therefore also that same row of $\mathbf{H}$. This would give a value of zero for each coordinate of this volume point. Using the coordinate delta instead of the coordinates themselves means that this zero value is now a zero displacement rather than zero coordinate. This is appropriate, as the point should not move if it is beyond the support radii of all base set nodes. Alternatively to Equation (66), the displacement of the volume points can be evaluated via two other equivalent methods. Firstly, $\boldsymbol{\alpha}$ can be constructed to create an interpolation of displacements rather than positions:

$$\boldsymbol{\alpha}^i_\Delta = \boldsymbol{\Phi}^{-1} \Delta \mathbf{B}^i \tag{67}$$

The displacement of the volume nodes can then be evaluated from Equation (64):

$$\Delta \mathbf{V}^i = \boldsymbol{\Psi} \boldsymbol{\alpha}^i_\Delta \tag{68}$$

Lastly, $\boldsymbol{\alpha}^i_\Delta$ can also be found through finding the difference between $\boldsymbol{\alpha}$ evaluated from the original and moved base node positions:

$$\boldsymbol{\alpha}^i_\Delta = \boldsymbol{\alpha}^i_{\text{moved}} - \boldsymbol{\alpha}^i_{\text{initial}} \tag{69}$$

As these approaches are all equivalent, they will not be distinguished throughout the rest of this thesis.

4.2 Variable Base Node Support Radii

A specific selection of the base set support radii was used for the forward flight simulations in this thesis, in order to both quicken the calculations as well as improve the deformation quality and ease of application of the RBF methods. This subsection develops an understanding of the link between the properties of the base set and the system conditioning, before explaining the advantages of this new method for choosing the support radii.

The condition number of Φ is related to the proximity of the base nodes to each other relative to the support radii. To illustrate this, Figure 23 shows a contour of the condition number of Φ according to where a third base node is placed. There are two preexisting base nodes at $(-0.5, 0)$ and at $(0.5, 0)$, which are marked by red crosses. The base nodes have a support radius of 1 and the contour is scaled with a base 10 logarithm.

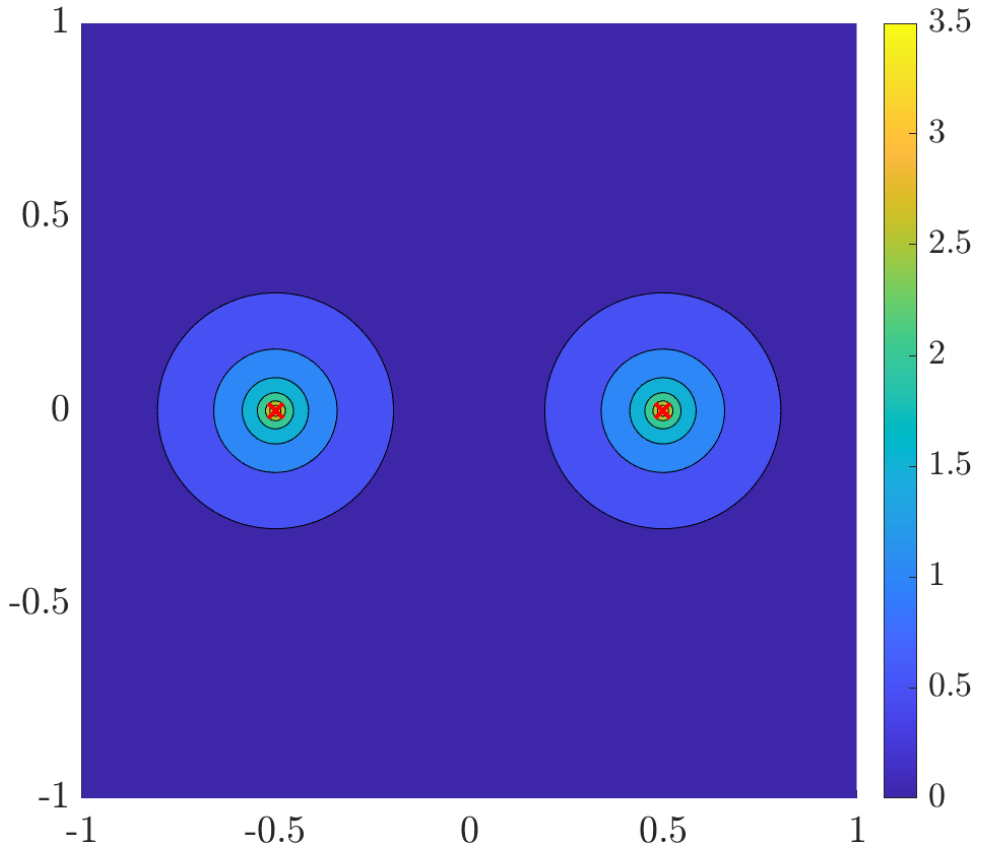


Figure 23: $\log_{10}$ of the condition number of $\mathbf{\Phi}$ according to location of a third base node, given two base nodes at the red crosses

It can be seen that there are large peaks in the resulting condition number at the locations of the preexisting nodes. This can be verified mathematically for the simple cases of two base nodes. A two node system gives:

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_{1,2} \\ \phi_{1,2} & 1 \end{pmatrix} \tag{70}$$

assuming that the support radii are constant and an RBF with $\phi(0) = 1$ is chosen. The identity for the condition number of a normal matrix is:

$$\varkappa(A) = \frac{|\iota_{\max}|}{|\iota_{\min}|} \tag{71}$$

where ι symbolises an eigenvalue of the matrix. The eigenvalues of the matrix in Equation (70) are $(\phi_{1,2} - 1)^2$ and $(\phi_{1,2} + 1)^2$. Since $\phi_{1,2}$ is on the interval $[0, 1]$, $(\phi_{1,2} - 1)^2$ is the eigenvalue with the smallest modulus. As a result, the condition number can be expressed as:

$$\varkappa(\mathbf{\Phi}) = \frac{(\phi_{1,2} + 1)^2}{(\phi_{1,2} - 1)^2} \tag{72}$$

As one base node approaches the other, $\|\mathbf{x} - \mathbf{x}_0\| \to 0$, and correspondingly $\xi \to 0$ and $\phi_{1,2} \to 1$. Therefore, $\varkappa(\mathbf{\Phi}) \to \infty$. This demonstrates that the actual

coordinates are unimportant; it is the scaled distance between the points that is critical. Furthermore, let us consider the behaviour of the system when the two points are coincident. When the two base nodes are identical, the base node to base node influence matrix is:

$$\mathbf{\Phi} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \tag{73}$$

This matrix has no inverse. This uninvertibility also extends to larger RBF systems with two coincident points. The general structure of $\mathbf{\Phi}$ with coincident points can be written as:

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_{1,2} & \dots & \phi_{1,x} & \dots & \phi_{1,x} & \dots & \phi_{1,N_b} \\ \phi_{1,2} & 1 & \dots & \phi_{2,x} & \dots & \phi_{2,x} & \dots & \phi_{2,N_b} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_{1,x} & \phi_{2,x} & \dots & 1 & \dots & 1 & \dots & \phi_{x,N_b} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_{1,x} & \phi_{2,x} & \dots & 1 & \dots & 1 & \dots & \phi_{x,N_b} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_{1,N_b} & \phi_{2,N_b} & \dots & \phi_{x,N_b} & \dots & \phi_{x,N_b} & \dots & 1 \end{pmatrix} \tag{74}$$

where there are two equal rows and columns somewhere within this matrix corresponding to the coincident points. As these two columns and rows are the same, the matrix has linearly dependent columns/rows, and hence also has no inverse. This uninvertibility with coincident points intuitively leads to the conclusion that two points being located near each other will also result in a poorly conditioned matrix, regardless of the locations of the other points. As $\mathbf{\Phi}$ in Equation (74) matrix is uninvertible, its determinant must be zero, and correspondingly one of the eigenvalues as well. If this eigenvalue were tracked as the two points approach each other, one would expect to see the eigenvalue approach zero smoothly, rather than with a discontinuity. Correspondingly, this would mean the denominator of Equation (71) approaches zero, and the condition number tends to infinity.

This behaviour of the eigenvalues is demonstrated in Figure 24. Figure 24 shows the eigenvalues of the $\mathbf{\Phi}$ matrix for a 1 dimensional system consisting of base nodes at $-0.5, 0, 0.5$ and a fourth point that takes values between -1 and 1. The coordinate of this moving point is plotted along the horizontal axis, with the eigenvalues on the vertical axis. It can be seen that the eigenvalues smoothly approach zero as the static and moving base nodes approach each other.

The intuition of growing condition number with converging base nodes is further supported for more complex systems with similar condition contour plots to Figure 23. Figure 25 displays two more example cases. On the left, the example shows a layout of a crude representation of two rotor blades. On the right, base nodes are arranged in a circle.

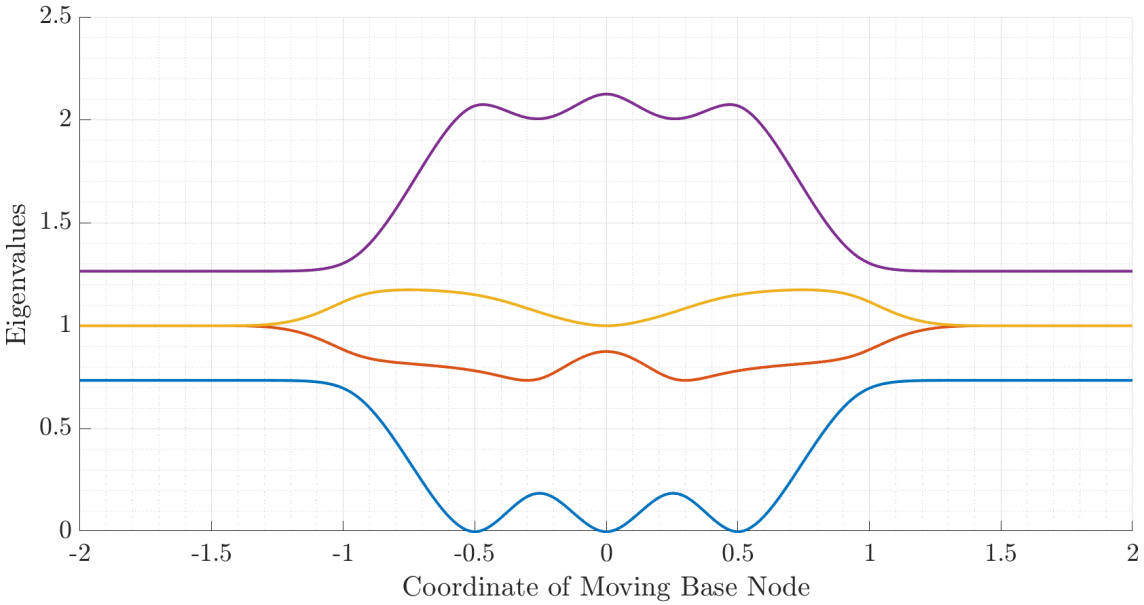


Figure 24: Eigenvalues of $\mathbf{\Phi}$ matrix with base nodes overlapping at -0.5, 0 and 0.5

Further examples can be found in Appendix A, showing a number of combinations of node locations and support radii. Across all the examples, there is a clear trend that the condition number increases as the support-radius-scaled distance decreases, tending towards infinity.

This discussion of the condition of $\mathbf{\Phi}$ relates to the point selection and setting of support radii. Considering the ARC rotor used in this thesis, it can be shown that the geometry of the rotor roots and hub will likely lead to poorly-conditioned systems. Assuming a traditional multiscale implementation of RBF mesh motion with a fixed support radius for the base set, this support radius must be chosen to accommodate the maximum displacement expected in the simulation, Δ_{max}. Specifically, Δ_{max} is the lower bound for r, otherwise the mesh will move into areas that do not get moved. Above this limit, there are many different values used across the literature. Table 11 shows a compilation of simulations which provided support radii or publications with recommendations for support radii. In this table, "mac" is the mean aerodynamic chord, C is a constant, r_{ref} is the support radius used in the article and the ϕ columns contain evaluations of the RBF between a point on the blade root inline with the centre of rotation and another point. For ϕ_{hub}, this is a point on the hub inline with the centre of rotation, for $\phi_{\text{next root}}$ this is the corresponding point on one of the blades ahead or behind the first blade, for $\phi_{\text{hub far}}$ this is the point on the opposite side of the hub, and for $\phi_{\text{far root}}$ it is the corresponding point on the opposite blade. The geometric data of the ARC blade dimensions defined in Figure 3, and the hub extends $0.53m$ in the direction of the rotation axis. A representative coning angle of 3.2° is used, which results in a maximum $\Delta = 0.408m$.

In view of Table 11, it can be seen that many of the support radii choices lead to large ϕ values at the near side of the hub. This even extends to the far side of the hub if the semispan is used as the support radius. Considering the clustering

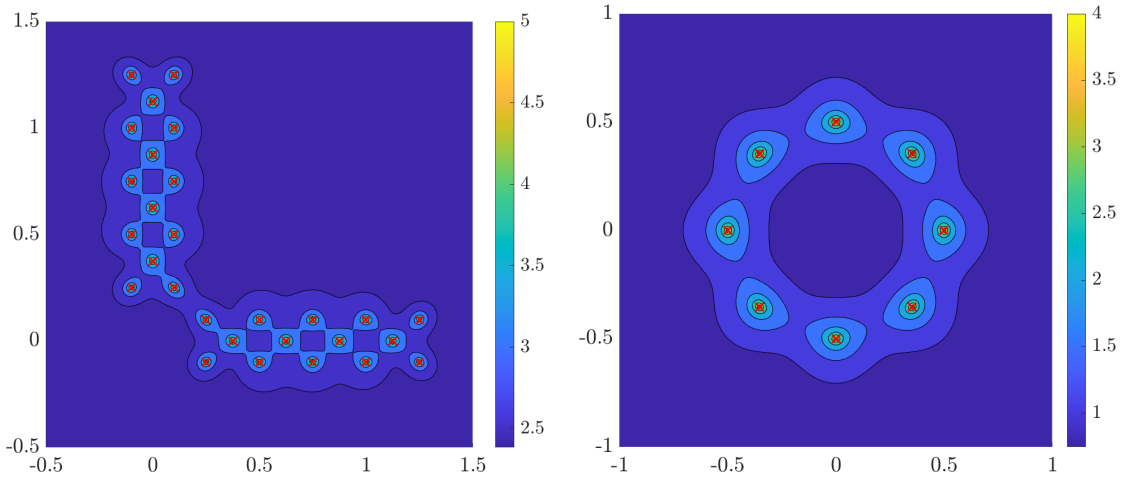


Figure 25: Examples of the condition number of $\mathbf{\Phi}$ peaking when a node in the base set approaches another base node, marked with red crosses. Condition number scaled by $\log_{10}$

Table 11: Evaluation of ϕ between the blade root centre and various locations using support radii from literature

Author	**Ref**	r_{ref}	**Condition**	r	ϕ_{hub}	$\phi_{\text{next root}}$	$\phi_{\text{hub far}}$	$\phi_{\text{far root}}$
Kedward et al.	[2]	$4c$	30° rotation aerofoil	2.13	0.707	0.0514	0.0267	0.00014
Kedward et al.	[2]	semispan	2m deflection wing	7.32	0.965	0.756	0.722	0.590
Rendall and Allen	[76]	$2c$	Rotor demo	1.07	0.288	0	0	0
Rendall and Allen	[76]	$5c$	Rotor demo	2.67	0.794	0.159	0.113	0.0179
Rendall and Allen	[77]	$10m$	4.8m deflection wing	0.851	0.143	0	0	0
Rendall and Allen	[77]	$50m$	8.5m deflection	2.40	0.756	0.102	0.0647	0.00424
Rendall and Allen	[68]	2 mac	4.5m deflection wing	1.04	0.269	0	0	0
Rendall and Allen	[68]	$C\Delta_{\max}$	Guideline	2.04	0.687	0.0371	0.0172	$5.21 \cdot 10^{-6}$
Rendall and Allen	[68]	$\Delta_{\max}$	Minimum value	0.408	0	0	0	0

of points on the four blade roots and on the hub, there would be a large number of points with large ϕ values, and therefore an ill-conditioned system should be expected as per the exploration of the matrix conditioning above.

There are a few possibilities for addressing this issue. These include removing the base nodes from the hub or reducing the base node density in this area. However, despite the multiscale method guaranteeing the position of all surface nodes, the effect on the near surface mesh would be noticeable. An example of this is shown in Figure 26, which shows a section of blade boundary layer mesh using the standard multiscale RBF method from Section 7.3. This section of the mesh shows an undulating nature, displaying how the differing refinement node support radii produce uneven meshes in the absence of sufficient deformation influence from base nodes.

Another solution is to allocate the support radii of the base nodes so that there

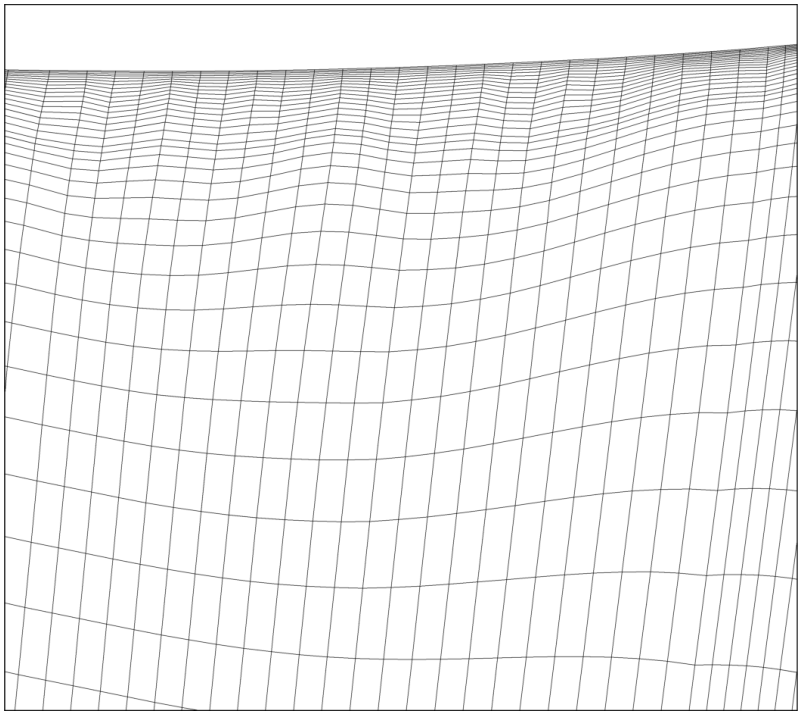


Figure 26: Demonstration of distorted a boundary layer mesh due to an inadequate number of local base nodes

is no overlap in the hub region. As an example, in the forward flight simulations in this thesis, the radii were selected as the minimum of three distances:

- The distance from the base node to the plane $x = y$
- The distance from the base node to the plane $x = -y$
- The minimum distance from the base node to any point on the hub

To demonstrate the effectiveness of using variable support radii, another $\mathbf{\Phi}$ condition contour plot was produced. Figure 27 is equivalent to the left subfigure of Figure 25, except the support radii are equal to the distance from the origin, rather than a constant value of 1. When comparing the figures, it is clear that the sensitivity of the condition number to adding points in the "blade root" region is noticeably reduced.

This approach has benefits beyond just the reduction of the condition number of $\mathbf{\Phi}$. Firstly, the rotor movement cannot reach the hub nodes. Therefore, there is no need to include them in the mesh movement algorithm as either base or refinement nodes. This significantly reduces the number of nodes in the algorithm, benefiting both the conditioning, as explained above, but also performance, with a smaller system being solved. Reducing the system size is important, as in Section 4.4 it will be shown that the computing cost grows superlinearly with system size. With this approach, the mesh near the hub is close to unchanged. The greatest influences on this region will come from the base nodes, which have their support radii determined

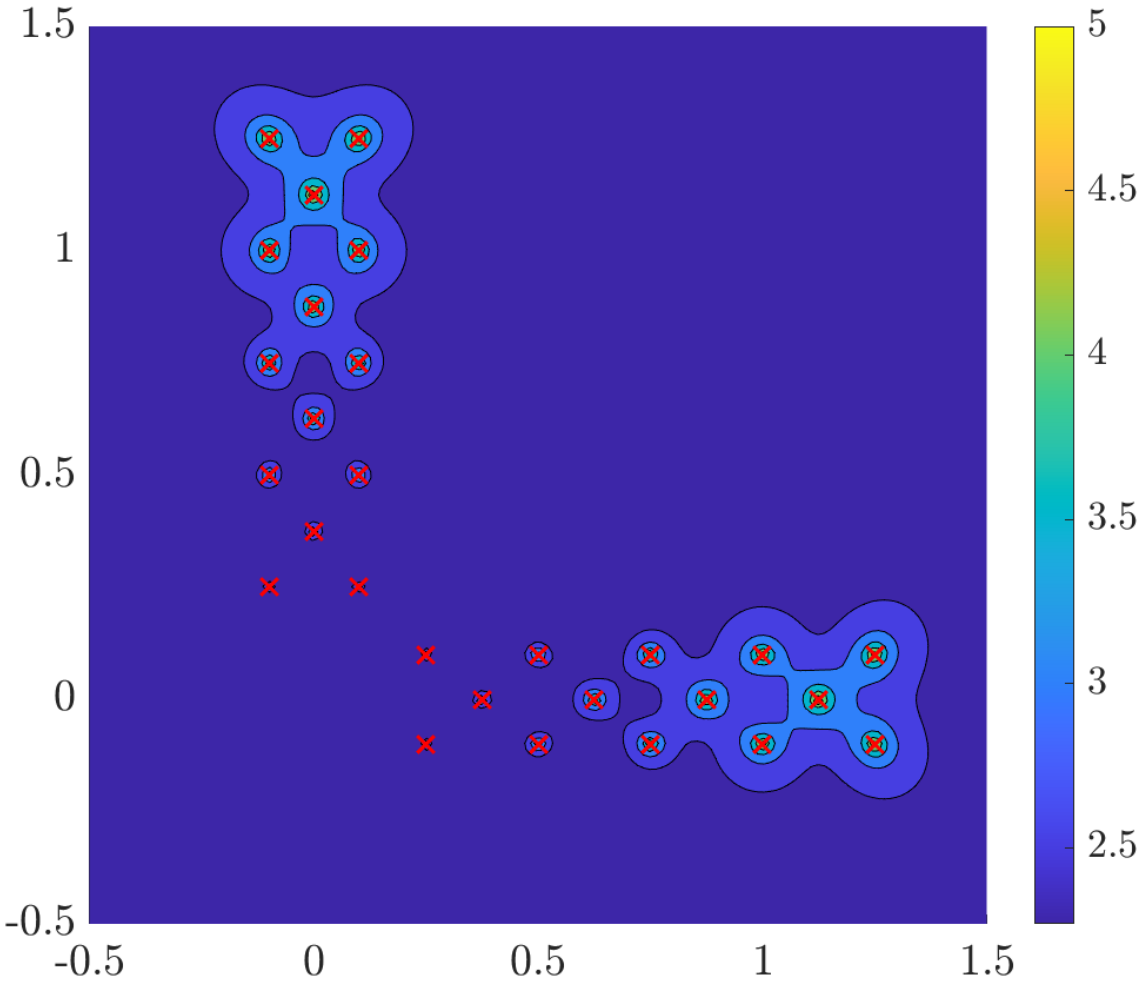


Figure 27: Contour of the condition number of $\mathbf{\Phi}$, variable r, simplified blade shape. Condition number scaled by $\log_{10}$

as the distance to the hub. Considering the ϕ value as the strength of the movement influence, ϕ is 0.1 from $\xi \approx 0.584$. This is correspondingly 58.4% of the distance from the base node to the hub. Similarly, ϕ is under 0.01 from $\xi \approx 0.778$. With such small values of ϕ close to the hub, the mesh deformation in this region is negligible.

This approach is applicable to simulations of rotating bladed objects in general, with the appropriate planes chosen instead of $x = \pm y$ for rotors with a different number of blades. Care should be taken that the support radii are appropriately large for the relevant blade motions.

As the deformation regions for each blade are nonoverlapping, this approach also allows the possibility of separating the simulation into four separate mesh motion systems, one for each blade. This approach may be a promising ground for large computational speed-ups due to the increased paralleliseability and smaller RBF systems, which could be traded off against higher base node counts per blade to improve mesh deformation quality. This approach is further discussed in Section 8.2.

4.3 An Iterative Greedy Point Selection Algorithm for Multiscale Algorithms

This thesis introduces an RBF point selection algorithm that applies the greedy algorithm for base node selection discussed in Section 3.7 to the multiscale implementation of Kedward et al. [2], additionally incorporating iterative methods in order to minimise the computational cost. This development combines the increased performance, mesh quality and exact surface specification of the multiscale method with the close to optimal point selection of greedy algorithms. This maximises the mesh quality for a given number of base set nodes, or correspondingly, minimises the number of base set nodes required for a given mesh quality, which improves algorithm performance during timestepping. The union of the greedy and multiscale methods would lead to extremely high preprocessing costs if not for the use of iterative algorithms for the greedy point selection.

4.4 Iterative Methods

As discussed in Sections 3.8 and 3.9, the computing time required for greedy point selection algorithms scales unfavourably with $O(N_b^4)$, but can be greatly reduced by iterative methods. Due to the variable support radii and application of the multiscale method, the iterative processes previously presented in the literature had to be redeveloped and expanded. The methods presented here iterate the following variables of the greedy algorithm:

- Base set list $\mathbf{B}$
- List of refinement nodes $\mathbf{R}$
- Base node support radii
- Separation distances at specific stages of the surface preprocessor
- Base set to base set influence matrix $\mathbf{\Phi}_b$
- Inverse base set to base set influence matrix $\mathbf{\Phi}_b^{-1}$
- The base set to refinement set influence matrix $\mathbf{\Phi}_r$
- Part of the refinement set to refinement set influence matrix $\mathbf{L}$
- The volume node influence matrices $\mathbf{\Psi}_{b,v2}$, $\mathbf{\Psi}_{b,v3}$ and $\mathbf{\Psi}_{r,v3}$

The iteration of the node lists is simple, with the node selected by the greedy algorithm being moved from the refinement node list to the reduced base set. Similarly, with the base set support radii being added to each iteration and each element being constant across iterations, this can also simply be added to each iteration.

The iterative algorithms for the rest of the objects are detailed below, with the algorithms presented in order of execution within each iteration. The execution order of these algorithms is largely arbitrary, although the surface preprocessor must be run first.

4.4.1 Surface Preprocessor

With the surface preprocessing step needing to be rerun each iteration of the greedy algorithm, there are opportunities for iterative methods. The first step of the surface preprocessor is calculating the separation distances of each of the refinement nodes to the base set nodes. However, as the base set is only ever added to, it is not necessary to recompute the distances to the base set nodes from previous iterations. Therefore, these separation distances $\mathbf{\Xi}_{b_i}$ can be calculated as:

$$\mathbf{\Xi}_{b_i} = \min(\mathbf{\Xi}_{b_{i-1}}, \mathbf{\Xi}_{b_{\text{new}}}) \tag{75}$$

Where $\mathbf{\Xi}_{b_{i-1}}$ is the list of separation distances from the previous iteration and $\mathbf{\Xi}_{b_{\text{new}}}$ is the separation distance from the refinement nodes to the new point added to the base set. There is an additional step that must be taken prior to the above operation, namely that the separation distance corresponding to the refinement node moved to the base set must be removed from $\mathbf{\Xi}_{b_{i-1}}$.

Additionally, a number of preprocessor steps may be condensed into one. This begins with comparing the separation distances to the new base node to the final separation distances (and hence support radii) assigned to each of the refinement nodes at the end of the previous greedy algorithm step. As the surface preprocessor only depends on the minimum distance between points, if the newly added base node is beyond the current separation distances of the refinement nodes, it will have no impact on the support radii and sorting of these nodes. In more detail, one can construct such an algorithm:

1. Compare the separation distances of the refinement radii $\mathbf{\Xi}_{i-1}$ to the separation distance to the newly added base node $\mathbf{\Xi}_{b_{\text{new}}}$.
2. Find the first refinement node that is closer to the new base node than it's separation distance $\mathbf{r}_n : \mathbf{\Xi}_{b_{\text{new}}}(n) < \mathbf{\Xi}_{i-1}(n)$. The index of the last unchanged refinement node will be N_p.
3. Copy the first N_p old refinement nodes and support radii up to the active set
4. Calculate the separation distance from the inactive set to the refinement nodes just added
5. Update the separation distance of the inactive set as $\min(\mathbf{\Xi}_b, \mathbf{\Xi}_{\text{ref added}})$

Compared to running the preprocessor from the beginning, for all refinement nodes, this saves updating the separation distances for the points copied from the previous step of the greedy algorithm.

4.4.2 Base to Base Node Influence Matrix $\mathbf{\Phi}_b$

The iteration of $\mathbf{\Phi}_b$ is also quite straightforward, as can be seen from its structure. The structure of $\mathbf{\Phi}_{b_i}$, $\mathbf{\Phi}_b$ at iteration i of the greedy algorithm, can be described as:

$$\mathbf{\Phi}_{b_i} = \begin{pmatrix} \mathbf{\Phi}_{b_{i-1}} & \boldsymbol{\curlyvee}_1 \\ \boldsymbol{\curlyvee}_2 & \phi_{i,i} \end{pmatrix} \tag{76}$$

Where $\mathbf{\Phi}_{i-1}$ is the base set to base set influence matrix from the previous iteration of the greedy algorithm. $\boldsymbol{\curlyvee}_1$ is the column vector $[\phi_{1,N_b}, \phi_{2,N_b}, \ldots, \phi_{N_b-1,N_b}]^T$, the influence of the new base node on the previous base set nodes. $\boldsymbol{\curlyvee}_2$ is the row vector $[\phi_{N_b,1}, \phi_{N_b,2}, \ldots, \phi_{N_b,N_b-1}]$, the influences of the previous base set nodes on the new one. r_1 must be calculated since the support radius of the added point changes (due to it switching from being a refinement node to a base node). However, since r_2 contains the influences of the previous support nodes on the new one, this vector can be taken from the row of $\mathbf{\Phi}_{r_{i-1}}$ that corresponds to the new reduced base set node. The refinement node where the greatest error criterion was calculated will be represented by $\mathbf{r}_E$, and the index of its location in $\mathbf{R}$ is i_E, so that $\mathbf{R}(i_E) = \mathbf{r}_E$. As the Wedland's C2 function is employed as the RBF for the simulations performed in this thesis, $\phi(0)$ evaluates to 1, and hence all values of $\phi_{i,i}$ are 1. This allows $\mathbf{\Phi}_i$ to be simplified to:

$$\mathbf{\Phi}_{b_i} = \begin{pmatrix} \mathbf{\Phi}_{b_{i-1}} & \boldsymbol{\curlyvee}_1 \\ \boldsymbol{\curlyvee}_2 & 1 \end{pmatrix} \tag{77}$$

which saves one calculation of ϕ each iteration. This is also valid for all other RBFs with $\phi(0) = 1$.

4.4.3 Inverse Base to Base Node Influence Matrix $\mathbf{\Phi}_b^{-1}$

The iteration of $\mathbf{\Phi}_b^{-1}$ is more complex than the previous variables, but due to the high cost of matrix inversion, it also brings a significant performance benefit. Applying the identity from Lu and Shiou [88]:

$$\begin{aligned} &\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \\ &\quad = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix} \\ &\quad = \begin{bmatrix} (A - BD^{-1}C)^{-1} & -(A - BD^{-1}C)^{-1}BD^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & D^{-1} + D^{-1}C(A - BD^{-1}C)^{-1}BD^{-1} \end{bmatrix} \end{aligned} \tag{78}$$

to the structure of $\mathbf{\Phi}_i$ defined in Equation (76) allows us to define the inverse of $\mathbf{\Phi}_i$ as:

$$\mathbf{\Phi}_{b_i}^{-1} = \begin{pmatrix} \mathbf{\Phi}_{b_{i-1}}^{-1} + \lambda \mathbf{\Phi}_{b_{i-1}}^{-1} \boldsymbol{\curlyvee}_1 \boldsymbol{\curlyvee}_2 \mathbf{\Phi}_{b_{i-1}}^{-1} & -\lambda \mathbf{\Phi}_{b_{i-1}}^{-1} \boldsymbol{\curlyvee}_1 \\ -\lambda \boldsymbol{\curlyvee}_2 \mathbf{\Phi}_{b_{i-1}}^{-1} & \lambda \end{pmatrix} \tag{79}$$

Where $\boldsymbol{\curlyvee}_1$ and $\boldsymbol{\curlyvee}_2$ have the same definitions as above, and λ is a scalar value defined as:

$$\lambda = \frac{1}{\phi_{i,i} - \boldsymbol{\curlyvee}_2 \boldsymbol{\Phi}_{b_{i-1}}^{-1} \boldsymbol{\curlyvee}_1} \tag{80}$$

This method follows a similar construction to the methods of Zhao et al. and Skala [3, 89]. However, due to the variable support radii, the symmetry of base set to base set matrix cannot be used to simplify the mathematics. Similarly to Equation (77), the definition of λ can be simplified for RBFs with $\phi(0) = 0$:

$$\lambda = \frac{1}{1 - \boldsymbol{\curlyvee}_2 \boldsymbol{\Phi}_{b_{i-1}}^{-1} \boldsymbol{\curlyvee}_1} \tag{81}$$

4.4.4 Base to refinement Node Influence Matrix $\boldsymbol{\Phi}_r$

An iterative method was also constructed for $\boldsymbol{\Phi}_r$. First, it can be seen that $\boldsymbol{\Phi}_{r_i}$ has the structure:

$$\boldsymbol{\Phi}_{r_i} = [\boldsymbol{\Phi}_{r_{\text{old b}}}, \boldsymbol{\Phi}_{r_{\text{new b}}}] \tag{82}$$

where $\boldsymbol{\Phi}_{r_{\text{old b}}}$ is the influence of $\mathbf{B}_{i-1}$ on the refinement nodes, and $\boldsymbol{\Phi}_{r_{\text{new b}}}$ is the influence of the new base node on the refinement nodes. It should be noted that $\boldsymbol{\Phi}_{r_{\text{old b}}}$ should also have the row i_E removed. As the base set radii are constant across iterations, the influence of the previous base set on a given node is unchanged. However, the order of these points may be changed by the refinement node reordering in the surface preprocessing step. Therefore, a refinement node order vector $\mathbf{o}_r$ can be constructed, with element i equal to the position of refinement node i at the previous step. With this, $\boldsymbol{\Phi}_{r_{\text{old b}}}$ can be formed by reordering the rows of $\boldsymbol{\Phi}_{r_{i-1}}$ according to $\mathbf{o}_r$: $\boldsymbol{\Phi}_{r_{\text{old b}}} = \boldsymbol{\Phi}_{r_{i-1}}(\mathbf{o}_r, :)$. This sampling of $\boldsymbol{\Phi}_{r_{i-1}}$ also covers the removal of the row i_E.

4.4.5 Refinement to Refinement Node Influence Matrix L

Some of $\mathbf{L}$ may also be constructed using $\mathbf{o}_r$ and $\mathbf{L}_{i-1}$, however, more care must be taken due to the refinement node radii being nonconstant. Furthermore, the storage method used for $\mathbf{L}$ also changes how the matrix can be iterated. For this reason, this section will cover iteration algorithms for two methods for storing $\mathbf{L}$: storing the whole structure and storage using Compressed Sparse Row (CSR) storage. The former may be useful for smaller implementations where the number of refinement nodes is not too large and the implementation of sparse storage methods is not worth the additional coding complexity. CSR storage is used as an example for sparse storage due to the ease of accessing rows of $\mathbf{L}$, which is useful for solving the coefficients for the refinement nodes with $\mathbf{L}\boldsymbol{\beta} = \boldsymbol{\Delta}_r$. In addition to $\mathbf{o}_r$, this procedure requires the vectors of refinement node support radii at this iteration and the previous one: $\boldsymbol{\rho}_{r_i}$ and $\boldsymbol{\rho}_{r_{i-1}}$. Additionally, these procedures require two additional inputs. The first of these is $\mathbf{p}$, a vector of the new positions of each refinement node. Specifically $\mathbf{p}(i)$ is the index of $\mathbf{R}_{i-1}(i)$ in $\mathbf{R}_i$. This can be easily produced by the surface processor by executing the following each iteration:

$$\mathbf{p}(i_M) = i \tag{83}$$

Where i_M is the index of the point with maximum separation distance during iteration i. The second requirement is $\mathbf{o}_{r_n}$. This is the new refinement node order like $\mathbf{o}_r$, but with all values above i_E reduced by one. This is equivalent to the position of refinement node i on the previous step, if node i_E were already removed.

The iterative method for $\mathbf{L}$ is based on two observations. The first is that the evaluations of ϕ are only dependant on the distance between two points and the support radius associated with the influencing node. Secondly, although the refinement nodes are reordered, and moved to the base node list, their coordinates themselves do not change. With these in mind, it can be concluded that if the support radius associated with a given refinement node does not change, the values of ϕ associated with that point can be copied from $\mathbf{L}_{i-1}$ to $\mathbf{L}_i$.

Iterating a $\mathbf{L}$ with full matrix storage requires the steps seen in Algorithm 1. This algorithm compares the refinement node support radii from this iteration to the reordered radii vector from the previous iteration. Two lists of refinement nodes which can be copied are generated, with the lists referring to both the previous refinement node order, i_{copy}, and to the current order $i_{copy,new}$. The rows that can be copied from $\mathbf{L}$ are all rows that don't correspond to node i_E. The function $evalPhi$ should return a matrix with the RBF evaluations for all combinations of the influence points with their support radii, which are function arguments one and two, and the influenced points, argument three.

Algorithm 1 IterateFullL

$\boldsymbol{\rho}_{r_{i-1},o} \leftarrow \boldsymbol{\rho}_{i-1}(\mathbf{o}_r)$
$matches \leftarrow \boldsymbol{\rho}_{r_i} = \boldsymbol{\rho}_{r_{i-1},o}$
$i_{copy} \leftarrow \mathbf{o}_r(matches)$
$i_{copyR} \leftarrow \mathbf{o}_r(\mathbf{o}_r \neq i_E)$
$i_{copy,temp} \leftarrow \mathbf{o}_{r_n}(matches)$
$i_{copy,new} \leftarrow \mathbf{p}(i_{copy,temp})$
refVec $\leftarrow 1:1:N_r$
$i_{calc} \leftarrow$ refVec$(\ matches)$
$\mathbf{L}_i(\text{refVec}, i_{copy,new}) = \mathbf{L}_{i-1}(i_{copyR}, i_{copy})$
Calculate columns i_{copyR} of $\mathbf{L}_i$
$\mathbf{L}_i(\mathbf{p}, i_{calc}) \leftarrow evalPhi(\mathbf{r}(i_{calc}, :), \boldsymbol{\rho}(i_{calc}), \mathbf{r}(\mathbf{p}, :))$

Before presenting the iterative algorithm for $\mathbf{L}$ in CSR format, a description of this format is necessary. The CSR format represents a sparse matrix using three vectors:

- a, containing the value of every nonzero element of the matrix
- ja, containing the corresponding column of every nonzero element of the matrix

- ia, containing the cumulative number of nonzero elements up to and including row i in $ia(i+1)$ (assuming zero-based indexing)

With $\mathbf{L}$ in CSR format, the iteration is more complicated due to not being able to manipulate $\mathbf{L}$ as a whole, but merely the nonzero elements of it. The iteration of $\mathbf{L}$ in CSR format shares a first common step to the iteration of $\mathbf{\Psi}_{r,v3}$: determining which columns of the objects can be copied from the previous iteration, and which columns have to be calculated anew. As the columns correspond to the refinement nodes, this is equivalent to determining which refinement nodes have not changed their influence on the surrounding points, which in turn means that the associated support radius has remained unchanged. The algorithm for generating this data is shown in Algorithm 2.

Algorithm 2 GenerateCopyCalcInds

$\boldsymbol{\rho}_{r_{i-1},o} \leftarrow \boldsymbol{\rho}_{i-1}(\mathbf{o}_r)$
$matches \leftarrow \boldsymbol{\rho}_{r_i} = \boldsymbol{\rho}_{r_{i-1},o}$
$copyLogicalVec \leftarrow false$
$copyLogicalVec(newRefOrder(matches)) \leftarrow true$
$calcInds \leftarrow refVec(notmatches)$

Due to the different ways the copies and calculations are dealt with in the following algorithms, the copyable and calculate indices are stored in different formats. The copyable indices are stored in a vector of length N_r+1 whose values are true for the indices of $\mathbf{L}_{i-1}$ and $\mathbf{\Psi}_{r,v3_{i-1}}$ that can be copied into the new objects. The rows to be calculated are stored as the indices of the new objects which are to be operated on. With this data known, the algorithm to iterate $\mathbf{L}$ is presented in Algorithm 3.

This algorithm loops over each row of $\mathbf{L}_i$ and extracts the corresponding row from $\mathbf{L}_i$. These values are then iterated over, checking if they are copyable and if they are within the lower triangular matrix structure, and if so, added to the new row vector. Once this is done, the calculate indices are iterated over, and added to the new row vector when the RBF is nonzero. This leads to a vector for the new row that is comprised of two subvectors which are ordered. The first subvector will be ordered as the copyable points are those with refinement radii, and hence separation distances, that are the same as last iteration. Considering that the structure preprocessor orders the points by separation distance, these points will be in the same order as the previous iteration. In the case of points that have the same separation distance, they should be ordered the same way, as the preprocessor will treat them in the same way as previously. The second subvector of the newly calculated points will be ordered as it is generated by iterating over relevant rows of $\mathbf{L}$ in order. Since the subvectors are both ordered, the fastest way to sort the full vector is to use a merge algorithm instead of using a full sorting algorithm. However, these merge algorithms should be modified to return the new order of the sorted vector, as this is needed to permute the $aThisRow$ vector correspondingly. An example of such an algorithm is provided in Algorithm 4.

Algorithm 3 iterL

for $ii = 1 : N_r$ **do**
  rowIndMin $\leftarrow ia(\mathbf{o}_r(ii)) + 1$
  rowIndMax $\leftarrow ia(\mathbf{o}_r(ii+1))$
  rowInds $\leftarrow rowIndMin : rowIndMax$
  $aTemp \leftarrow a(rowInds)$
  $jaTemp \leftarrow ja(rowInds)$
  Zero vectors tracking the new values in this row
  $n_{nzR} \leftarrow 0$
  $aThisRow \leftarrow []$
  $jaThisRow \leftarrow []$
  Check values from $\mathbf{L}_{i-1}$ and copy if appropriate
  for $jj = 1 : numel(jaTemp)$ **do**
    $thisJa \leftarrow jaTemp(jj)$
    if $copyLogicalVec(thisJa)$ **then**
      if $thisJa > i_E$ **then**
        $thisJa \leftarrow thisJa - 1$
      end if
      $thisJaNew \leftarrow newRefPositions(thisJa)$
      if $thisJaNew \leq ii$ **then**
        $n_{nzR} \leftarrow n_{nzR} + 1$
        $jThisRow(n_{nzR}) \leftarrow thisJaNew$
        $aThisRow(n_{nzR}) \leftarrow aTemp(jj)$
      end if
    end if
  end for
  Iterate over calc indices and add to $\mathbf{L}_i$ if appropriate
  $thisRef \leftarrow \mathbf{R}(ii, :)$
  for $jj = calcInds$ **do**
    if $jj \leq ii$ **then**
      $thisR = \boldsymbol{\rho}(jj)$
      $dTemp = \|thisRef - \mathbf{R}(jj, :)\|$
      if $dTemp < thisR$ **then**
        $n_{nzR} = n_{nzR} + 1$
        $d = dTemp/thisR$
        $aThisRow(n_{nzR}) = \phi(d)$
        $jaThisRow(n_{nzR}) = jj$
      end if
    end if
  end for
  $n_{nzP} \leftarrow n_{nz}$
  $n_{nz} \leftarrow n_{nz} + n_{nzR}$
  $[jaThisRow, sortOrder] = mergeSortedSubvectors(jaThisRow)$
  $jaNew(n_{nzP} : n_{nz}) \leftarrow jaThisRow$
  $aNew(n_{nzP} : n_{nz}) \leftarrow aThisRow(sortOrder)$
  $iaNew(ii + 1) \leftarrow n_{nz}$
end for

Algorithm 4 MergeSortedSubvectors

procedure MERGE(n,vec,iStartV2,indices,vecOrdered,order)
  $i \leftarrow 1$ ▷ Merged vector index
  $iCount1 \leftarrow 1$ ▷ Index for first subvector
  $iCount2 \leftarrow iStartV2$ ▷ Index for second subvector
  while $iCount1 < iStartV2$ **AND** $iCount2 \leq n$ **do**
    if $vec(iCount1) \leq vec(iCount2)$ **then**
      $vecOrdered(i) \leftarrow vec(iCount1)$
      $order(i) \leftarrow indices(iCount1)$
      $iCount1 \leftarrow iCount1 + 1$
    else
      $vecOrdered(i) \leftarrow vec(iCount2)$
      $order(i) \leftarrow indices(iCount2)$
      $iCount2 \leftarrow iCount2 + 1$
    end if
    $i \leftarrow 1 + 1$
  end while
  At this point one vector is exhausted, move the remaining elements from the other vector to the sorted vector
  if $iCnt1 < iStartV2$ **then**
    while $iCnt1 < iStartV2$ **do**
      $vecOrdered(i) \leftarrow vec(iCount1)$
      $order(i) \leftarrow indices(iCount1)$
      $iCount1 \leftarrow iCount1 + 1$
      $i \leftarrow i + 1$
    end while
  else
    while $iCnt2 \leq n$ **do**
      $vecOrdered(i) \leftarrow vec(iCount2)$
      $order(i) \leftarrow indices(iCount2)$
      $iCount2 \leftarrow iCount2 + 1$
      $i \leftarrow i + 1$
    end while
  end if
end procedure

4.4.6 Base and Refinement Node to Volume Matrices $\boldsymbol{\Psi}$

The structures of the $\boldsymbol{\Psi}$ submatrices lend themselves to iteration as the volume nodes do not change each iteration. However, different methods are required for the base to volume node influence matrices $\boldsymbol{\Psi}_{b,v2}$ and $\boldsymbol{\Psi}_{b,v3}$, and for the refinement node to volume influence matrix $\boldsymbol{\Psi}_{r,v3}$. Similarly to above, this is because the base node list is only appended to, whereas the refinement nodes may change in order and in RBF radius. For the base node to volume node influence matrices, the method presented in Equation (84) works for both volume node categories:

$$\boldsymbol{\Psi}_{b,v_i} = \left[\boldsymbol{\Psi}_{b,v_{i-1}}, \boldsymbol{\Psi}_{b_{\text{new}},v}\right] \tag{84}$$

In Equation (84), $\boldsymbol{\Psi}_{b,v_{i-1}}$ is $\boldsymbol{\Psi}_{b,v}$ from the previous iteration, and $\boldsymbol{\Psi}_{b_{\text{new}},v}$ is the influence vector of the new base node on the volume points: $[\phi_{1,N_b}, \phi_{2,N_b}, \ldots, \phi_{N_v,N_b}]^T$. This method only has to calculate $\boldsymbol{\Psi}_{b_{\text{new}},v}$ each iteration of the iterative greedy algorithm.

As $\boldsymbol{\Psi}_{r,v3}$ is typically very sparse, a method for iterating this matrix with both full representations and using CSR storage will be presented. The iteration algorithm for a full storage of $\boldsymbol{\Psi}_{r,v3}$ is very similar to the algorithm to iterate the full storage $\mathbf{L}$., due to the matrices both tracking the influence of the refinement nodes on a group of points. Algorithm 5 shares the determining of the copyable columns, but is overall simpler than Algorithm 1 due to the row order being consistent across iterations.

Algorithm 5 IteratePsiB

$\boldsymbol{\rho}_{r_{i-1},o} \leftarrow \boldsymbol{\rho}_{i-1}(\mathbf{o}_r)$
$matches \leftarrow \boldsymbol{\rho}_{r_i} = \boldsymbol{\rho}_{r_{i-1},o}$
$i_{copy} \leftarrow \mathbf{o}_r(matches)$
$i_{copy,new} \leftarrow \mathbf{o}_{r_n}(matches)$
$i_{copy,new} \leftarrow \mathbf{p}(i_{copy,new})$
refVec $\leftarrow 1:1:N_r$
$i_{calc} \leftarrow$ refVec$(\ matches)$
$\boldsymbol{\Psi}_{b,v_i}(:, i_{copy,new}) = \boldsymbol{\Psi}_{b,v_{i-1}}(:, i_{copy})$
Calculate columns i_{calc} of $\boldsymbol{\Psi}_{b,v_i}$
$\boldsymbol{\Psi}_{b,v_i}(:, i_{calc}) \leftarrow evalPhi(\mathbf{r}(i_{calc}, :), \boldsymbol{\rho}(i_{calc}), \mathbf{V})$

Using the CSR format for storing $\boldsymbol{\Psi}_{r,v_3}$ also leads to an iterative algorithm very similar to that for CSR $\mathbf{L}$. The primary differences to Algorithm 3 are that the row order is consistent across iterations of the greedy algorithm, and that the matrix is not definitely lower triangular. Algorithm 6 shows the iteration of $\boldsymbol{\Psi}_{r,v_3}$, using the same merge algorithm as in the iteration of $\mathbf{L}$.

Algorithm 6 Iterate $\mathbf{\Psi}_{r,v_3}$

for $ii = 1 : N_v$ **do**
  rowIndMin $\leftarrow ia(ii) + 1$
  rowIndMax $\leftarrow ia(ii + 1)$
  rowInds $\leftarrow rowIndMin : rowIndMax$
  $aTemp \leftarrow a(rowInds)$
  $jaTemp \leftarrow ja(rowInds)$
  Zero vectors tracking the new values in this row
  $n_{nzR} \leftarrow 0$
  $aThisRow \leftarrow []$
  $jaThisRow \leftarrow []$
  Check values from $\mathbf{\Psi}_{r,v_3 i-1}$ and copy if appropriate
  for $jj = 1 : numel(jaTemp)$ **do**
    $thisJa \leftarrow jaTemp(jj)$
    if $copyLogicalVec(thisJa)$ **then**
      if $thisJa > i_E$ **then**
        $thisJa \leftarrow thisJa - 1$
      end if
      $thisJaNew \leftarrow newRefPositions(thisJa)$
      $n_{nzR} \leftarrow n_{nzR} + 1$
      $jThisRow(n_{nzR}) \leftarrow thisJaNew$
      $aThisRow(n_{nzR}) \leftarrow aTemp(jj)$
    end if
  end for
  Iterate over calc indices
  $thisVol \leftarrow \mathbf{V}_3(ii, :)$
  for $jj = calcInds$ **do**
    $thisR = \boldsymbol{\rho}(jj)$
    $dTemp = \|thisVol - \mathbf{R}(jj, :)\|$
    if $dTemp < thisR$ **then**
      $n_{nzR} = n_{nzR} + 1$
      $d = dTemp/thisR$
      $aThisRow(n_{nzR}) = \phi(d)$
      $jaThisRow(n_{nzR}) = jj$
    end if
  end for
  $n_{nzP} \leftarrow n_{nz}$
  $n_{nz} \leftarrow n_{nz} + n_{nzR}$
  $[jaThisRow, sortOrder] = mergeSortedSubvectors(jaThisRow)$
  $jaNew(n_{nzP} : n_{nz}) \leftarrow jaThisRow$
  $aNew(n_{nzP} : n_{nz}) \leftarrow aThisRow(sortOrder)$
  $iaNew(ii + 1) \leftarrow n_{nz}$
end for

4.5 Cost Analysis of Iterative Methods

With the iterative methods for a greedy multiscale algorithm presented, an analysis of computation savings is warranted. Similarly to Kedward et al. [2] and Zhao et al. [3], this will be shown through a comparison of the cost of each method using (a somewhat loose application of) big O notation. Within the context of computation, big O notation is most often used to represent the cost of algorithms as the input size becomes indefinitely large. For example, the cost of a sorting algorithm as the input vector becomes arbitrarily large. More formally, for functions $f(n), g(n) : \mathbb{N} \to \mathbb{R}^+$, it is said that $f(n)$ is $O\left(g(n)\right)$ if there exist $c, n_0 \in \mathbb{N}^+$ such that:

$$f(n) \leq cg(n) \tag{85}$$

for all $n \geq n_0$ [93]. Big O notation places a bound on the asymptotic growth of $f(n)$, saying that, within a constant multiplicative factor, $f(n)$ grows no faster than $g(n)$ [94]. In the context of computational complexity analysis, $f(n)$ is typically a function representing the number of operations or steps taken by an algorithm or program. This is a proxy for the algorithm's runtime, which is highly dependant on many factors including the code implementation, software and hardware.

One rule of Big O notation is that a $C \cdot O(x), C \in \mathbb{R}^+$ algorithm is still $O(x)$, as the definition accounts for constant multipliers (the value of c in Equation (85) simply changes). As a result, it is usually pertinent to consider the most expensive operations within the innermost loop of an algorithm. In this case, keeping in line with the work of Kedward et al. [2] and Zhao et al. [3], each evaluation of the RBF function between two points can be considered as one operation. Similarly, in the context of the preprocessor, one distance calculation will be counted as one operation. Furthermore, as per Zhao et al. [3], it will be assumed that there is no cost to use a mathematical object from a previous iteration. For example, if appending to a matrix from the previous iteration, only the cost of calculating the new elements for the current iteration will be considered.

As will be demonstrated, matrix inversions are the dominant step when considering the Big O analysis of both the iterative and noniterative greedy multiscale methods. Therefore, each iterative algorithm shall be analysed separately, with a comparison to the corresponding noniterative methods. The iteration of simple objects, such as lists of nodes or their corresponding RBF radii, will not be covered as they are computationally insignificant.

4.5.1 Surface Preprocessor

According to Kedward et al. [2], the asymptotic cost of the surface preprocessor can be calculated as:

$$\sum_{i=1}^{N_s-1} (N_s - 1) + O(N_s) + O(N_r) = \frac{N_s(N_s-1)}{2} + O(N_s) + O(N_r) = O(N_s^2) \tag{86}$$

this consisting of $\sum_{i=1}^{N_s-1}(N_s-i)$ operations for updating the separation distances each step, $O(N_s)$ operations for reordering the nodes, and $O(N_r)$ operations for producing the vector of refinement node support radii. As $N_s > N_r$, the $O(N_s^2)$ distance calculations are the dominant term, with the calculations of $O(N_s)$ and $O(N_r)$ being insignificant as the system size grows. The distance calculations will also be the dominant costs for the iterative algorithms, and hence these will be the focus of the following analysis.

Costing the iterative algorithm for the surface preprocessor involves analysing the cost of calculating the distances to the base set and then the iteration over the refinement nodes. Let us first analyse the savings from only calculating the separation distance to the new base node at each iteration. Instead of the sum seen in Equation (86), the separation distances to the base set can be evaluated with:

$$N_r + \sum_{i=1}^{N_r-1}(N_r - i) = \sum_{i=0}^{N_r-1}(N_r - i) \tag{87}$$

distance calculations. There are N_r evaluations needed for the distances to $N_{b_{\text{new}}}$, and the sum accounts for the evaluations over the rest of the preprocessor. The number of additional calculations done by the noniterative method can be determined through rearranging the sum in Equation (86):

$$\sum_{i=1}^{N_s-1}(N_s - i) = \sum_{i=N_b}^{N_s-1}(N_s - i) + \sum_{i=1}^{N_b-1}(N_s - i) = \sum_{i=0}^{Nr}(N_r - i) + \sum_{i=1}^{N_b-1}(N_s - i) \tag{88}$$

The second reformulation uses the property that all surface nodes are either refinement or base nodes, and hence $N_s = N_r + N_b$. This property does not strictly have to be true, but is natural when using the multiscale method, as it is designed to allow exact surface positioning without unreasonable computational cost. Comparing Equations (87) and (88) shows that $\sum_{i=1}^{N_b-1}(N_s - i) = 1/2(N_b - 1)(2N_s - N_b)$ calculations are saved by this part of the iterative methods. In application, some of this will be offset by finding the minimum of $\boldsymbol{\Xi}_{b_{i-1}}$ and $\boldsymbol{\Xi}_{b_{new}}$, as well as passing $\boldsymbol{\Xi}_{b_{i-1}}$ to the preprocessor as an additional argument.

Calculating the savings due to finding the separation distances to the first N_p refinement points before entering the preprocessor loop is more difficult, as there is no way to generally predetermine N_p. This parameter is dependant on the geometry of the case, as well as which points are chosen as base nodes, and hence the deformation or movement used to select the base nodes. However, an algebraic comparison can still be made for the reduction in steps for the merged iterations. The number of distance calculations for the merged steps is:

$$N_p\,(N_r - N_p) + \sum_{i=1}^{N_r-N_p-1}(N_r - N_p - i) \tag{89}$$

With a similar strategy to Equation (88), the number of distance calculations in the iteration stage of the preprocessor can be shown to be:

$$\sum_{i=1}^{N_r-1}(N_r-i)=\sum_{i=1}^{N_p}(N_r-i)+\sum_{i=1}^{N_r-N_p-1}(N_r-N_p-i) \tag{90}$$

Correspondingly, the difference in the methods (iterative minus merged) is equivalent to:

$$\sum_{i=1}^{N_p}(N_r-i)-N_p(N_r-N_p)=\frac{N_p}{2}(N_p-1) \tag{91}$$

This shows no difference for $N_p = 0$ or 1, as would be expected, but quadratically growing savings for N_p above these values. The total number of operations when combining the base node separation distance skipping and iteration merging is:

$$N_r+N_p(N_r-N_p)+\sum_{i=1}^{N_r-N_p-1}(N_r-N_p-i)=\frac{1}{2}\left(N_r^2-N_p^2+N_r+N_p\right)=O(N_r^2) \tag{92}$$

It can be seen from Equations 86 and 92 that the iterative approaches do not change the asymptotic behaviour of the algorithms, but do change the variable with which they scale. This is reflected in the total costs of the algorithms over the complete base set selection. The total number of distance calculations expected for a noniterative greedy multiscale approach would be:

$$\sum_{i=1}^{N_b}\frac{N_s(N_s-1)}{2}=\frac{1}{2}N_bN_s(N_s-1)=O(N_bN_s^2) \tag{93}$$

This assumes that N_s is constant over the iterations. The asymptotic cost of the algorithm grows linearly in N_b and quadratically in N_s. In contrast, it is not possible to get a definite expression for the total cost of using the iterative method, as N_p is not constant over the iterations and has no way of being determined. However, the total distance calculations for an iterative method with only the base node distance skipping can be evaluated. Using the substitution $N_r = N_s - N_b$, this total cost is:

$$\frac{1}{6}N_b\left(3N_s^2-3N_sN_b+N_b^2-1\right) \tag{94}$$

This expression is more complex to analyse than Equation (93), because the cost as a function of only N_b is not monotonic increasing, and the behaviour of this function must be looked at within the appropriate interval $1 \leq N_b \leq N_s$. As such, big O notation is not appropriate for describing the algorithm cost. However, the two cost functions intersect at $N_b = 0, 1, 3N_s - 1$, and thus there are no roots in $1 \leq N_b \leq N_s$. It can be easily checked that the iterative algorithm costs less than the noniterative algorithm at $N_b = N_s$, and therefore is always less costly than the noniterative algorithm. To help understand the scale of the savings, the total distance calculations for both algorithms with $N_s = 100\,000$ can be seen in Figure 28.

However, this cost comparison still does not account for N_p. Whilst it is impossible to provide a formula for N_p, some typical distributions of N_p over a range of scenarios are provided in Figure 29. These cases include:

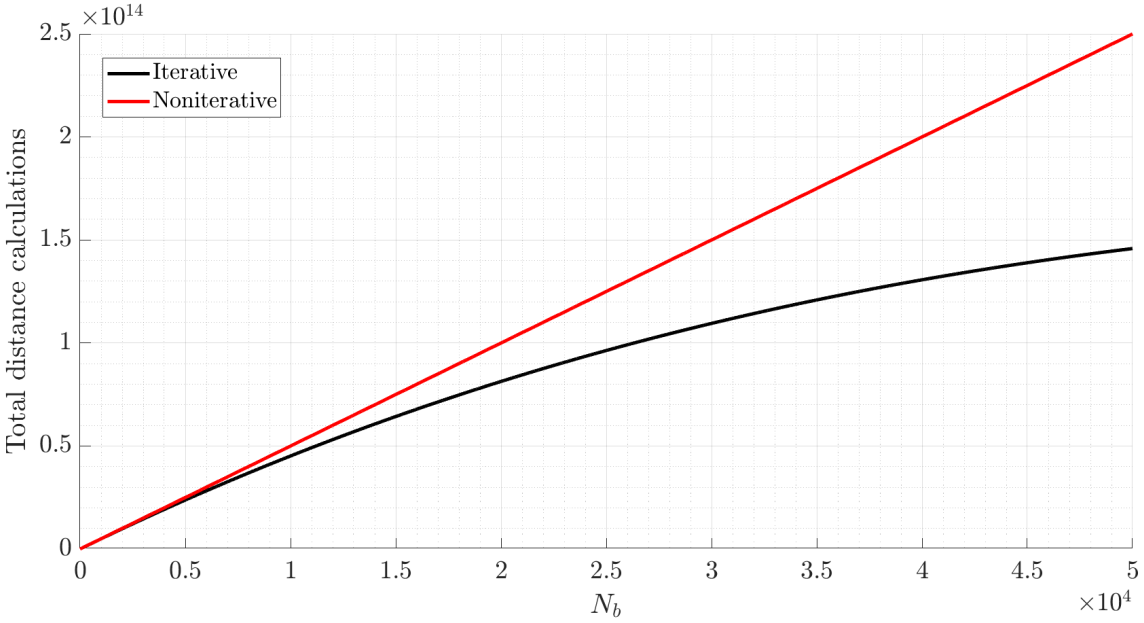


Figure 28: Example costs for the preprocessor with $N_s = 100\ 000$

- The mesh used for forward flight with the maximum pitch and coning angles experienced in the trim state for an advance ratio of 0.3
- The forward flight mesh with doubled pitch and coning angles
- A low-resolution version of the forward flight mesh with the maximum pitch and coning angles experienced at an advance ratio of 0.3
- A simplified mesh consisting of four rotor blades of diamond aerofoils. The mesh contains a total of 57 624 nodes. A depiction of this mesh can be found in Appendix B. The mesh is deformed with the maximum pitch and coning angles as for the forward flight simulations at an advance ratio of 0.3.

Figure 29 displays N_p as a fraction of N_r at each iteration. This information is displayed as a moving mean over 50 iterations. Figure 29 shows that N_p is definitely large enough to affect computational times. The behaviour between the lower resolution test meshes and the full resolution meshes is distinct, with the test meshes reaching significantly higher N_p fractions. However, when the iterations are scaled by the number of surface points, reflecting the fraction of the surface nodes selected as base nodes, the distributions are more similar, with the diamond aerofoil and two full resolution meshes all having approximately $N_p/N_r = 0.08$ at $N_{b_i}/N_s = 0.01$. This suggests that the N_p fraction may correlate with the fraction of the total surface nodes selected. Considering the $1/2N_p(N_p - 1)$ calculations saved each iteration in light of these distributions, the theoretical savings of the iterative preprocessor should grow with N_b even faster than suggested by Figure 28.

4.5.2 Base Set to Base Set Influence Matrix $\mathbf{\Phi}_b$

Iteration of $\mathbf{\Phi}_b$ consists of two steps: appending the appropriate row from $\mathbf{\Phi}_r$ to the bottom of $\mathbf{\Phi}_b$, and calculating the row corresponding to the influences of the

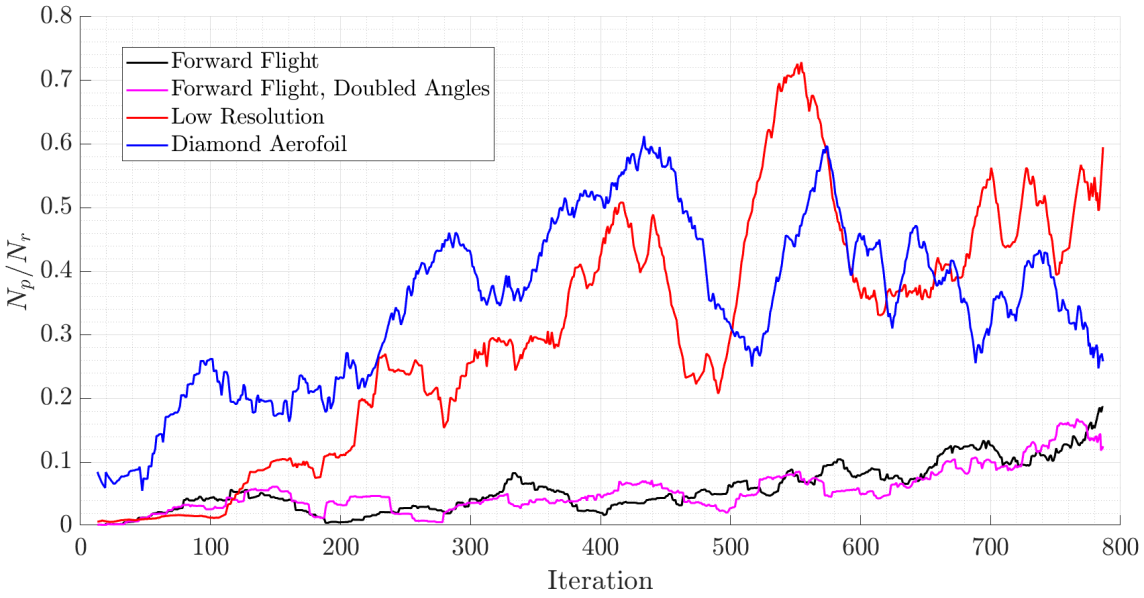


Figure 29: Example distributions of N_p as a fraction of N_r

new base set node. It is assumed that appending the N_b values to $\mathbf{\Phi}_r$ takes $O(N_b)$ operations, in line with the storage of the refinement radii in Kedward et al. [2]. Calculating the N_b new influences requires $O(N_b)$ operations. With one operation for calculating $\mathbf{\Phi}_b$ for the first base node, the sum of operations over the N_b chosen base set nodes is:

$$1 + \sum_{N_{b_i}=2}^{N_b} N_{b_i} = \frac{N_b(N_b+1)}{2} = O(N_b^2) \tag{95}$$

For a noniterative method, the N_b^2 elements of $\mathbf{\Phi}_r$ must be calculated each step, with total cost:

$$\sum_{N_{b_i}=1}^{N_b} N_{b_i}^2 = \frac{1}{6} N_b \left(N_b+1\right)\left(2N_b+1\right) = O(N_b^3) \tag{96}$$

Hence, the iterative methods reduce the asymptotic cost by an order.

4.5.3 Inverse Base to Base Node Influence Matrix $\mathbf{\Phi}_b^{-1}$

As per Kedward et al. [2] and Zhao et al. [3], the cost of inverting a matrix is $O(N^3)$. This leads to a total number of operations as per Kedward et al. [2]:

$$\sum_{N_{b_i}=1}^{N_b} O(N_{b_i}^3) = O(N_b^4) \tag{97}$$

The iterative method requires $O(N^2)$ operations per inversion according to Skala [89]. This means the total cost for the inversions of $\mathbf{\Phi}_b$ is:

$$\sum_{N_{b_i}=1}^{N_b} O(N_{b_i}^2) = O(N_b^3) \tag{98}$$

4.5.4 Base to Refinement Node Influence Matrix $\mathbf{\Phi}_r$

$\mathbf{\Phi}_r$ has a size N_b by N_r, therefore requiring $N_b N_r$ operations to construct noniteratively. Considering that all surface points must either be base or refinement nodes, we can take advantage of the property $N_s = N_b + N_r$ to express this cost as $N_b(N_s - N_b)$. Across the iterations of the greedy algorithm, the total cost equates to

$$\sum_{N_{b_i}=1}^{N_b} N_{b_i}(N_s - N_{b_i}) = \frac{1}{6} N_b \left(N_b + 1\right) \left(3N_s - 2N_b - 1\right) \tag{99}$$

As this expression is not monotonic increasing when N_s is taken as constant, and $N_b \leq N_s$, this cannot be given a valid Big O cost. However, when considering the cost as a function of N_s, the asymptotic growth is linear.

The iterative greedy multiscale method only has to evaluate ϕ for the new column corresponding to the most recently added base node, all other data is taken from $\mathbf{\Phi}_{r_{i-1}}$. This column is of length N_r, or correspondingly $N_s - N_b$. This means that the total cost of this method over all the iterations is:

$$\sum_{N_{b_i}=1}^{N_b} (N_s - N_{b_i}) = \frac{1}{2} N_b \left(2N_s - N_b - 1\right) \tag{100}$$

This expression is again linear in N_s but nonmonotonic in N_b. However, when comparing the expressions as a function of N_b, the iterative evaluations are always equal to or less than the noniterative method. Similarly to the preprocessor, the expressions are equal at $N_b = 0, 1, {}^3\!/_2 N_s - 1$, and the noniterative method is easily proven to require fewer calculations at $N_b = N_s$, and hence is smaller across all scenarios. An example comparison of the total costs can be seen in Figure 30, with $N_s = 100\ 000$ and costs shown across $1 \leq N_b \leq 50\ 000$.

4.5.5 Refinement to Refinement Node Influence Matrix L

Considering the lower triangular structure of $\mathbf{L}$, the number of ϕ evaluations required for the noniterative method is:

$$\sum_{j=1}^{N_r} j = \frac{1}{2} N_r (N_r + 1) = O\left(N_r^2\right) \tag{101}$$

Estimating the algorithmic complexity of the iterative method is difficult due to the operations being dependant on both how many elements need to be sorted in each loop of the algorithm, and how many elements need to be calculated in total. These are both ultimately a function of the refinement node locations, the

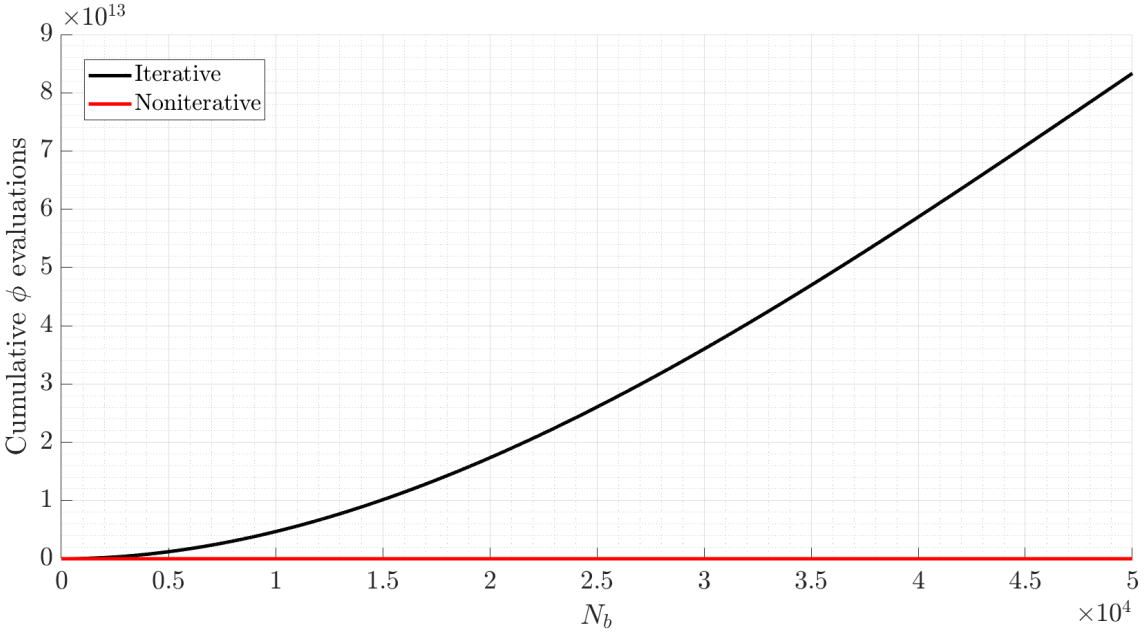


Figure 30: Example costs for the iteration of $\mathbf{\Phi}_r$ with $N_s = 100\ 000$

refinement node radii, and the change in the radii between the previous and current iterations. Considering that these are in turn a function of the geometry of the case being studied, as well as the deformation or movement of the relevant bodies, these are impossible to investigate analytically across all possible cases. However, a worst-case scenario can be examined. In this case, the are no elements that can be copied from $\mathbf{L}_{i-1}$ to $\mathbf{L}_i$, and furthermore $\mathbf{L}_i$ has no zero elements, meaning that each sort operates on the maximum number of elements. In reality, this scenario is extremely unlikely, as all refinement nodes would have to be within the RBF radius of all previous refinement nodes.

In analysing the worst-case scenario, the most expensive part of the algorithm's loop must be identified. Performing the calculate loop N_r times involves j evaluations of ϕ each iteration of the loop, where j is the row of $\mathbf{L}_i$ being calculated. As stated in Section 4.4.5, Algorithm 4 is a simple merge routine with the addition of returning the order of the sorted vector with regards to the input vector. A merge routine has a cost of $O(n)$ [94]. As this algorithm just adds a second assignment, to the order vector for every assignment to the sorted vector, this merely doubles the number of operations. As multiplication by constants do not change the asymptotic cost of an algorithm, Algorithm 4 is also $O(n)$. Similarly, with costs of $O(n)$ for both the RBF evaluations and the sorting, the cost per each row is also $O(n)$. This gives a total Big O cost the same as the noniterative algorithm as per Equation (101). If instead of Algorithm 4, a common $O(n \log n)$ sorting algorithm such as Heapsort, Quicksort or Mergesort is used, the worst-case cost of the iterative algorithm does rise above the noniterative methods by a very small margin. Details of this cost analysis can be found in Appendix C.

Similar to the analysis of the N_p distributions of the preprocessor, the number of

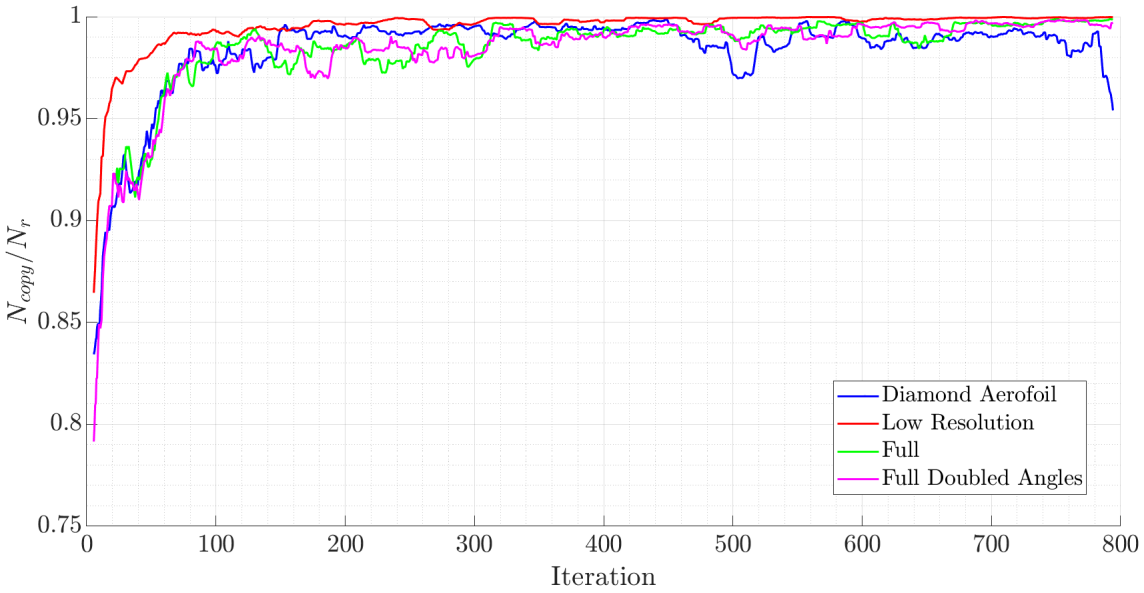


Figure 31: Distribution of copyable refinement influences as a fraction of N_r

refinement nodes that are copyable can be analysed for the same range of scenarios, as shown in Figure 31. It can be seen here that the proportion of rows that are copyable is very high, even in the early iterations. Although this will not affect the big O growth of the algorithm, as the row vector sorting cost is unchanged, this will reduce the real cost of the algorithm.

4.5.6 Volume Node Influence Matrices $\mathbf{\Psi}_{b,v}$ and $\mathbf{\Psi}_{r,v}$

As $\mathbf{\Psi}_{b,v}$ contains the evaluation of ϕ between each volume and base node, it contains a total of $N_v N_b$ evaluations of ϕ. Resultingly, producing this matrix noniteratively requires $N_v N_b$ operations. Summed over the iterations of the greedy algorithm, the total operations can be evaluated:

$$\sum_{N_{b_i}=1}^{N_b} N_v N_{b_i} = \frac{1}{2} N_v N_b (N_b + 1) = O(N_v N_b^2) \tag{102}$$

Conversely, an iterative production of $\mathbf{\Psi}_{b,v}$ only requires evaluating the influences of the new base node on the volume nodes, which is only N_v operations. Over the selection of all the base set, this totals:

$$\sum_{N_{b_i}=1}^{N_b} N_v = N_v N_b = O(N_v N_b) \tag{103}$$

Whilst both the iterative and noniterative methods asymptotically scale linearly with N_v, the iterative method scales linearly with N_b instead of quadratically.

The noniterative calculation of $\mathbf{\Psi}_{r,v3}$ involves $N_v N_r$ evaluations of ϕ, and hence is $O(N_v N_r)$. Similarly to the iteration of $\mathbf{L}$, the savings from iterating a CSR representation of $\mathbf{\Psi}_{r,v3}$ cannot be analytically determined across all cases. However, the

worst-case scenario can again be calculated. This worst-case scenario again entails having to recompute a fully nonzero matrix. In this case, there are N_v applications of the merge algorithm to N_r points, giving a cost of $O(N_v N_r)$. Similarly to iterating $\mathbf{L}$, the asymptotic cost of the worst-case scenario is the same as the noniterative method. However, the N_{copy} distributions are again relevant, which can be seen in Figure 31. This again suggests that the real cost of the iterative methods will scale much more favourably than the worst case.

4.5.7 Overview of Computational Savings

The iterative algorithms can be broken up into two groups: ones with system size, geometry and movement dependant costs, and ones whose costs are only dependant on the size of the system and number of base nodes selected. The iteration of $\mathbf{\Phi}_b$ and $\mathbf{\Phi}_b^{-1}$ are seen to have asymptotic costs of a degree lower than the noniterative methods, when considering scaling relative to the number of base nodes chosen. The other iterative methods are more complex to analyse over the total of the greedy algorithms due to the costings not lending themselves to big O analysis, or due to parts of the algorithms having no general algebraic form. The iteration of the surface preprocessor can be shown to definitely perform fewer distance calculations, with the savings increasing as a higher proportion of surface points are chosen as base nodes. The iteration of $\mathbf{\Psi}_r$ was shown algebraically to save evaluations of ϕ over the valid range of N_b. $\mathbf{L}$ and $\mathbf{\Psi}_{r,v3}$ do not have calculable big O costs, but expressions can be found for the worst possible cases. In this case, they are of the same order as the noniterative methods.

As the per-step costs of the surface preprocessor and the $\mathbf{\Phi}_r$, $\mathbf{L}$ and $\mathbf{\Psi}$ matrices can be expected to drop as N_{b_i} increases, the cost of the whole system will grow with the largest order of the remaining algorithms. This is the iteration of $\mathbf{\Phi}_b^{-1}$. Correspondingly, this is $O(n^4)$ for a noniterative method and $O(n^3)$ for the iterative method. However, due to the system sizes being finite and the number of indefinite costs, a true analysis of the relative speeds of these methods should be evaluated through experiment. These results are presented in Section 7.4.

4.6 Error Criterion

The greedy algorithms discussed in Section 3.7 used an error criterion based on the difference between the ideal moving boundary location and that generated by the reduced base set in order to select the point or points to be added to the reduced base set each iteration. Considering the multiscale method uses all surface points as either base set or refinement nodes, the location of the moving boundary is exact and there is no surface error, regardless of the number of bases nodes. Therefore, the iterative, greedy, multiscale method requires a different error criterion in order to select base nodes. Considering the surface is correctly defined, another desirable behaviour of the mesh deformation is that the boundary layer mesh quality is maintained post-deformation. Under this condition, it is preferable that the near-surface

mesh moves with the surface as closely as possible, for example, moving rigidly for simple translations or rotations. This method also maintains the original orthogonality of the mesh at the surface as well as possible. To try and achieve this result, the error criterion used in the iterative greedy multiscale method is the difference in location between near-surface mesh nodes and their "ideal" locations. These ideal locations are where the nodes would be if they were moved by the same conditions as the surface rather than by the RBFs. In choosing these nodes, it does not suffice to simply take the first layer of the mesh from the moving boundary. The rationale for this can be seen in Figure 32, where the differences between volume points rotated with the surface and volume points moved by the RBFs are compared as distance from the surface increases.

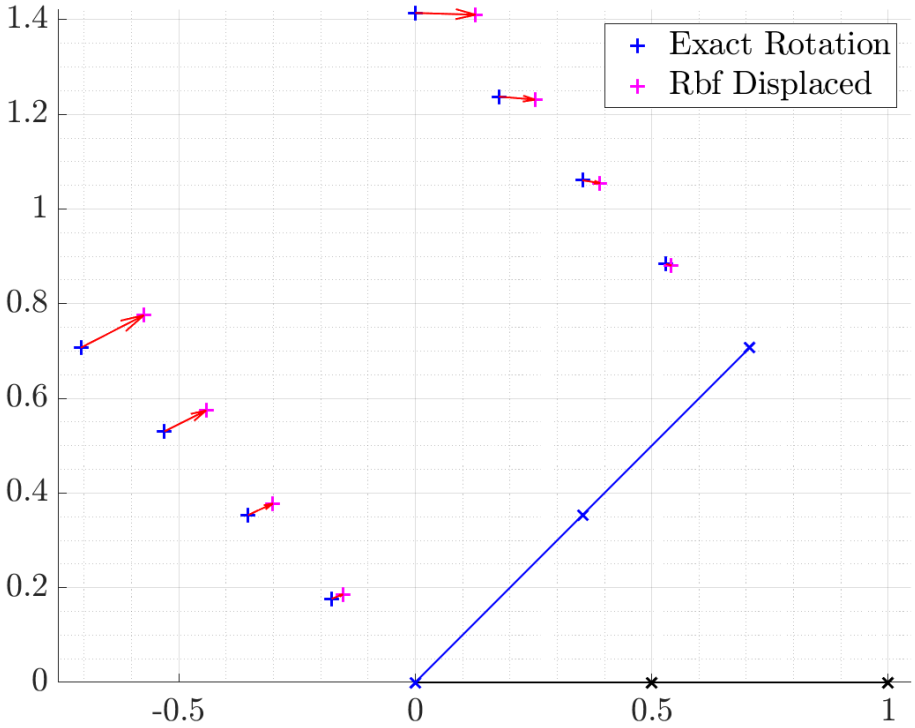


Figure 32: Demonstration of increasing difference between rigid movement and RBF deformation with distance from moving surface

Figure 32 shows that the difference between rigid and RBF motion increases with increasing distance from the surface. This is due to the strength of the RBF dropping off with the distance from the base set. Therefore, if the distances to the first nodes from the boundary are not equal, the location differences will be skewed and more points will be chosen near regions with a larger boundary spacing. To combat this, the near-surface points are generated by projecting a line between the surface nodes and the corresponding nodes in the first layer of the mesh, and setting the surface adjacent points to be a consistent, small distance from the surface nodes along these lines. Mathematically, the new surface adjacent point $\boldsymbol{v}_n$ is set to be distance d from surface point $\boldsymbol{s}$, on the interval between $\boldsymbol{s}$ and the surface adjacent mesh node $\boldsymbol{v}_a$:

$$\boldsymbol{v}_n = \frac{\boldsymbol{v}_a - \boldsymbol{s}}{\|\boldsymbol{v}_a - \boldsymbol{s}\|} d + \boldsymbol{s} \tag{104}$$

This process is executed for every surface node - surface adjacent node pair. This is shown visually in Figure 33. The surface mesh nodes are shown as black pluses (+), the surface adjacent mesh nodes as red crosses (×), and the newly generated adjacent points as blue squares (■).

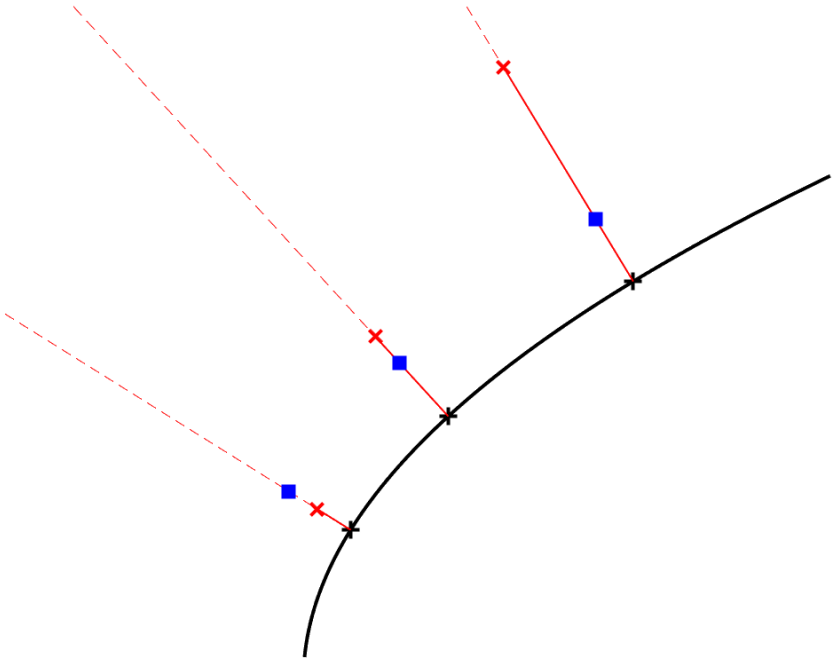


Figure 33: Production of surface adjacent nodes with equal distance from their corresponding surface nodes

5 CFD Solvers

This section covers the details of the two solvers used to perform the simulations in this thesis. Firstly, the details of the Actuator Surface Model will be presented, a hybrid CFD method for simulating rotors or wings in airflow. This includes the blade aerodynamics models and the coupling methods between the CFD solver and blade model. Furthermore, as the ASM uses OpenFOAM as a finite volume flow-solver, the numerical methods chosen within OpenFOAM are also presented. For the blade-resolved simulations, the high-order finite volume code Flamenco was used. The equations underpinning Flamenco will be presented, excluding the RBF mesh motion methods, which have already been detailed.

5.1 The Actuator Surface Model

The Actuator Surface Model is a hybrid computational fluid dynamics method that efficiently simulates time-accurate rotor loads and wakes, with the capability to be extended to contexts such as wind turbines and non-rotating aircraft wings. The model works by coupling a lower order model based on lifting line theory with finite volume Navier-Stokes solvers of OpenFOAM, capturing rotor-rotor and rotor-airwake interactions. The accuracy of this method lies between those expected of actuator disk modelling and blade resolved CFD methods. The basic idea of the ASM is to represent the blade through a group of momentum sources rather than meshing the actual blade geometry as per blade resolved simulations. This necessitates two-way coupling with flowfield velocities given from the CFD solver to the rotor model, and then momentum fluxes corresponding to the loads on the blade effected back on the CFD solver via momentum sources. The following subsections detail the models behind the ASM, covering the coupling methods and the internal aerodynamic systems. For a more detailed explanation of the various aspects of the ASM, as well as a presentation of numerous validation and demonstration cases, please see the work of Linton et al. [95, 96, 97, 98, 99] as well as Linton's doctoral thesis [100].

5.1.1 Blade Discretisation and Sectional Aerodynamics

The ASM has a number of parameters to accurately account for the rotor blade geometry and the blade's movements. The kinematics of the blades are modelled with a 1 degree of freedom hub and up to three 1 degree of freedom joints which can account for the blade flap, lead/lag motion and pitching. The positions and order of these joints can be specified to enable an accurate recreation of the blade motion. The blades themselves are geometrically defined using the following parameters:

- Aerofoil profile
- Chord length
- Twist angle

- Sweep angle
- Dihedral angle

These properties are specified at a number of sections along the blade. The aerofoil properties, chord and twist are assumed to vary linearly between each of these sections, whereas the sweep and dihedral are taken to be piecewise constant. Each of these sections are divided up into a number of panels, each of which has an associated bound vortex at the quarter chord, and a collocation point at 3/4 midspan where the effective angle of attack and panel loading is evaluated [101]. Across the blade there are typically 20-30 panels.

Each of these panels is assumed to behave as an isolated 2D aerofoil, with lift and drag calculated as a function of the flowfield and of time. The ASM requires that the 2D aerodynamics of each aerofoil section be known before a simulation. This is achieved through the specification of lift and drag coefficient tables for angles of attack from -180° to 180°. These tables should cover a range of Mach numbers that contain the expected Mach numbers experienced by each section of the blade. For these tables the Reynolds number is assumed to be proportional to the Mach number. When the aerodynamics of each section is calculated, the force coefficients are interpolated between the closest available angles of attack and Mach numbers. A correction for the drag force is then applied based on the Reynolds number of the actual flow and of the table coefficients.

5.1.2 Aerodynamic Models

Calculating the blade loads from just an interpolation of 2D aerofoil coefficients would be inadequate due to the numerous three-dimensional flow phenomena experienced by rotor blades, as well as the significant aerodynamic hysteresis caused by the blade motion and wake behaviour. This section presents the models used to account for these effects.

The application of Beddoes' unsteady aerodynamics model [102] in the ASM generates an equivalent angle of attack by applying two correction factors to the angle of attack. With the subscript n referring to the current timestep, this can be expressed as:

$$\alpha_{\text{eff},n} = \alpha_n - X_n - Y_n \tag{105}$$

These correction factors account for the hysteresis with regards to circulatory lift. The impulsive terms are neglected. X_n and Y_n are defined as:

$$X_n = X_{n-1} e^{-\frac{\Delta t}{T_1}} + A_1(\alpha_n - \alpha_{n-1}) e^{-\frac{\Delta t}{2T_1}} \tag{106}$$

$$Y_n = Y_{n-1} e^{-\frac{\Delta t}{T_2}} + A_2(\alpha_n - \alpha_{n-1}) e^{-\frac{\Delta t}{2T_2}} \tag{107}$$

where the $n-1$ subscript refers to the previous timestep. The values of the time delay constants T_1 and T_2 are given as:

$$T_1 = \frac{c}{2b_1(1-M^2)U_\infty}$$
$$T_2 = \frac{c}{2b_2(1-M^2)U_\infty} \tag{108}$$

and hence measure some distance travelled by the aerofoil relative to its chord, including a Mach scaling. The constants used in the above equations are set to $A_1 = 0.1$, $A_2 = 0.7$, $b_1 = 0.2$ and $b_2 = 0.65$ in the ASM implementation.

The equations for the yaw and sweep corrections are taken from the work of Johnson [9]. The effect of the yawed flow correction is primarily to align the viscous drag vector with the flow direction. It can be assumed that there is negligible effect on the lift that is generated. The yaw and sweep corrections are based on two measures: the yaw angle and the sweep angle. The yaw angle is defined as the angle between the incident flow and a line perpendicular to the reference quarter chord of the rotor blade. The sweep angle is the angle between the reference quarter chord and tangent to the actual quarter chord line. These can be seen in Figure 34 from Linton [101].

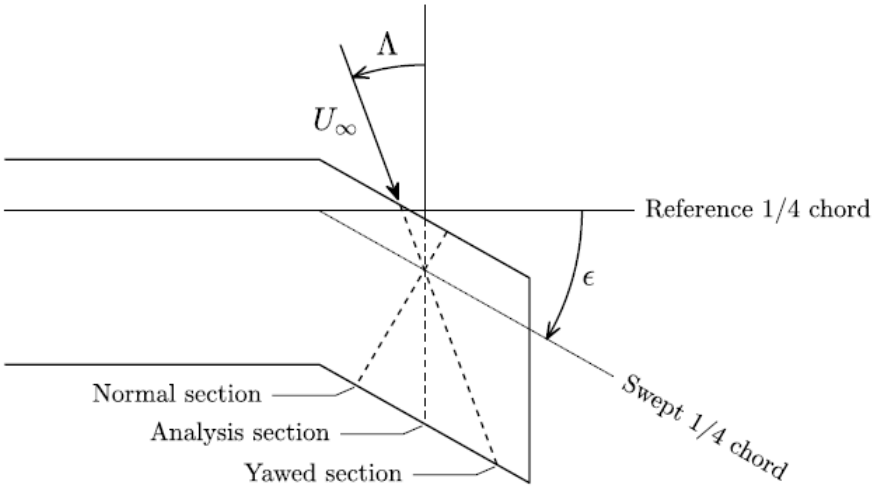


Figure 34: Definitions of the yawed and normal sections [101]

Firstly, a Mach number correction factor is defined as:

$$K_\epsilon = \frac{\cos(\Lambda + \epsilon)}{\cos\Lambda} \tag{109}$$

This is multiplied by the flow Mach number to find the normal Mach number, which determines the compressibility effects.

$$M_n = K_\epsilon M \tag{110}$$

Secondly, the Reynolds number effects are assumed to be dictated by the yawed Reynolds number:

$$Re_y = \frac{U_\infty c_y}{\nu} \tag{111}$$

Where c_y is the chord of the yawed section and is defined as:

$$c_y = c\frac{\cos(\epsilon)}{\cos\left(\Lambda + \epsilon\right)} \tag{112}$$

Using these definitions, the lift and chordwise drag coefficients can be calculated using the following equations:

$$\begin{aligned} \alpha_{e,l} &= \cos^2\left(\Lambda + \epsilon\right)\left(\frac{\alpha}{K_\epsilon} - \alpha_{zl}\right) + \alpha_{zl} \\ C_L &= \frac{1}{\cos^2\Lambda} C_{L,\mathrm{2D}}\left(\alpha_{e,l}, M_n, Re_y\right) \end{aligned} \tag{113}$$

$$\begin{aligned} \alpha_{e,d} &= \cos\left(\Lambda + \epsilon\right)\left(\frac{\alpha}{K_\epsilon} - \alpha_{zl}\right) \\ C_D &= \frac{1}{\cos\Lambda} C_{D,\mathrm{2D}}\left(\alpha_{e,d}, M_n, Re_y\right) \end{aligned} \tag{114}$$

where α_{zl} is the zero lift angle of attack, and the 2D subscript refers to the values from the 2D aerofoil coefficient tables.

5.1.3 Flow Sampling

The blade model would ideally sample the flow velocities at the locations of the aerofoil sections, however the velocities close to the blade are not useful. This is due to the momentum sources altering the local flow conditions, so that they are no longer representative of the incident flow on the aerofoil sections. Instead, the velocities are sampled at a distance c forward of the blade. This introduces a number of factors that have to be accounted for to acquire an accurate incident flow velocity. These include vorticity-induced velocity, wake downwash and downwash at the blade. Equation (115) displays the relationship between the corrected incident flow velocity and the correction factors [101]:

$$\boldsymbol{U}_\infty(t) = \boldsymbol{U}_g(\boldsymbol{p}_c, t) + \boldsymbol{U}_{\mathrm{CFD}}(\boldsymbol{p}_s, t_s) + \boldsymbol{U}_{in,w}(\boldsymbol{p}_c, t) - \boldsymbol{U}_{in,w}(\boldsymbol{p}_s, t_s) - \boldsymbol{U}_{in,b}(\boldsymbol{p}_s, t_s) \tag{115}$$

Here, $\boldsymbol{U}_g$ is the geometric velocity of the blade, $\boldsymbol{U}_{\mathrm{CFD}}$ is the velocity from the CFD solver sampled at the sample location, $\boldsymbol{U}_{in,w}(\boldsymbol{p}_c, t)$ is the downwash at the blade, $\boldsymbol{U}_{in,w}(\boldsymbol{p}_s, t_s)$ is the wake correction, and $\boldsymbol{U}_{in,b}$ is the correction for the bound vorticity. These terms are evaluated at either the sample point $\boldsymbol{p}_s$ or the blade collocation point $\boldsymbol{p}_c$, and either at the current time t or sample time $t_s = t - c/\Omega R$.

5.1.4 Momentum Distribution

For the calculated rotor loadings to be applied to the CFD flowfield, the one dimensional spanwise loading must be converted into a three dimensional momentum source distribution. These source distributions are governed by functions in the chordwise and blade normal directions, with the chordwise variable being labelled x_l and the blade normal variable labelled z_l. The integral of the product of these two functions over x_l and z_l should be equal to one. The chord normal distribution is the Gaussian function:

$$g_n(z_l) = \begin{cases} \frac{1}{c\varepsilon\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{z_l}{\varepsilon}\right)^2\right) & -3\varepsilon \leq z_l \leq 3\varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{116}$$

where ε is a Gaussian width parameter with value 0.1 and c is the local chord length. The chordwise distribution function model stems from a representation of bound circulation along a blade chord from Kim et al. [103] and is given by:

$$g_c(x_l) = \begin{cases} 0.4 + (3.04/0.25)x_l & 0 \leq x_l < 0.25 \\ 3.44 - (3.2/0.25)(x_l - 0.25) & 0.25 \leq x_l < 0.5 \\ 0.24 - (0.24/0.5)(x_l - 0.5) & 0.5 \leq x_l < 1 \\ 0 & \text{otherwise} \end{cases} \tag{117}$$

The calculation of the momentum source term is the product of these two functions with the spanwise loading distribution from the blade model $\boldsymbol{F}(y_l)$:

$$\boldsymbol{S}(x_l, y_l, z_l) = g_c(x_l) g_n(z_l) \boldsymbol{F}(y_l) \tag{118}$$

Where y_l is the spanwise coordinate. Figure 35 shows a two dimensional slice of this distribution through a blade section.

Considering that the mesh consists of a number of discrete volumes, this distribution is evaluated at each cell centre that finds itself within a bounding box that surrounds each blade. As such, the resolution of the mesh in this area must be sufficient to resolve these distributions with adequate fidelity. After evaluation, the total sum of the discrete momentum sources is checked against the momentum flux corresponding to the integrated loads from the blade model. If these do not agree, each of the discrete momentum sources are scaled so that the total momentum flux matches. However, this process is only valid for small scalings, which again limits the coarseness of the mesh in the rotor region. An edge length of $0.1c$ or less for the cells in the rotor region is generally sufficient.

5.1.5 Wake Model

As there is a delay between the sampling of the flowfield and that particular region reaching the blade, a history of corrected velocities is stored which can be time-interpolated to find the correct value [101]. As each blade section has an associated line of bound vorticity, the Biot-Savart law is used to calculate the corresponding

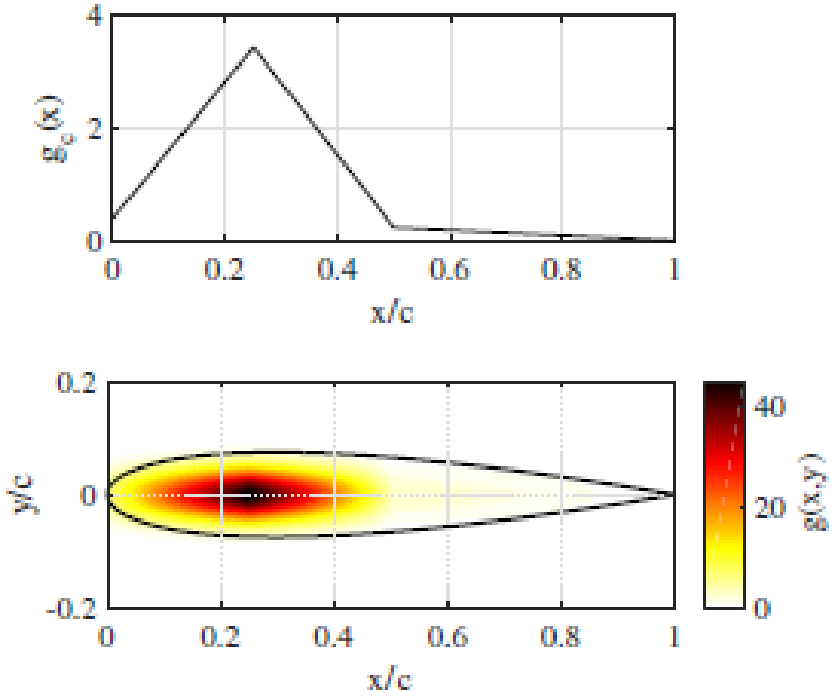


Figure 35: The chordwise momentum distribution function and a spanwise slice of the ASM momentum source distribution [101]

induced velocities and correct the velocities accordingly. Singularities are avoided by employing a vortex core model from Ramasamy and Lieshmann [104]. Lifting line models also inherently create downwash due to the vorticity that is trailed from the lifting surface. Due to the velocity being sampled in front of the blade, a wake model is included in the ASM model in order to calculate the effect of the wake downwash on the flow at the sample points. The final piece of the puzzle is the velocity of the blade itself relative to the reference frame used by the CFD solver.

The wake model employed by the ASM is used to compute the downwash and wake correction terms within the incident velocity calculations. These terms are determined by the contributions of the near-wake to a very large degree, and therefore the wake is only simulated up to an age of 45° [101]. The sampled velocities from the CFD solution capture the contributions of the mid- and far-wake effects. The wake model is comprised of vortex line segments which stem from the trailing edges of the blade panels [101]. Each of these segments begins and ends with a vortex marker, represented by massless Lagrangian particles. These markers are convected according to the CFD solution, this process being handled by the OpenFOAM solvers. A segment is released each time step. The circulation of each of these vortex markers is calculated according to Helmholtz's laws for the conservation of circulation. More specifically, the circulation is equal to the difference in the circulations of the bound vortices of the panels adjacent to the vortex marker release point. At the blade tip and root, where this is not applicable, the circulation is equal to the circulation of the bound vortices of the tip and root panel respectively. Circulation is assumed to vary linearly along the vortex line filaments.

5.1.6 OpenFOAM Numerics

The ASM simulations use the OpenFOAM solver pisoFoam which solves the incompressible Navier Stokes equations:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = -\nabla \mathbf{P} + \nu \nabla^{\mathbf{2}} \mathbf{U} + \mathbf{f} \tag{119}$$

and the continuity equation:

$$\nabla \cdot \mathbf{U} = \mathbf{0} \tag{120}$$

Here $\mathbf{U}$ is the velocity vector, $\mathbf{P}$ is the matrix of kinematic pressure (pressure normalised by a reference density), ν is the kinematic viscosity and $\mathbf{f}$ is a vector of source terms. pisoFoam solves these equations by using the Pressure-Implicit with Splitting of Operators (PISO) algorithm.

A combination of 1st and 2nd order methods are used for the spatial discretisation of the various terms.

Table 12: Numerical schemes for OpenFOAM in the ASM simulations

Term	**OpenFOAM Numerical Scheme**
∇	Gauss linear
$\nabla \cdot \mathbf{U}$	Gauss linear upwind
∇^2	Gauss linear limited corrected 0.333
Cell face interpolation	linear
Cell face normal gradient	limited corrected 0.333

Time stepping is performed using a backwards scheme:

$$\frac{\partial}{\partial t}(\varsigma) = \frac{1}{\Delta t}(\frac{3}{2}\varsigma^{n+1} - 2\varsigma^{n} + \frac{1}{2}\varsigma^{n-1}) \tag{121}$$

where the $n+1$ superscript represents the timestep to be calculated, n the current timestep, and $n-1$ the previous timestep of a generic variable ς.

5.2 Flamenco

The Flamenco flow solver is an in-house CFD code developed at The University of Sydney. First presented by Garcia et al. [105], Flamenco is a structured, curvilinear, Godunov-type finite volume code which solves the compressible Navier-Stokes equations. Flamenco employs a number of higher-order schemes, achieving 5th order accuracy in space for the inviscid terms, 2nd order accuracy in space for the viscous terms and 2nd order accuracy in time.

5.2.1 Governing Equations

Flamenco solves the compressible Navier-Stokes equations on a body-fitted curvilinear grid. The following derivations follow the work of Thomas and Lombard [106] and Pulliam and Steger [107]. This requires a transformation of the Navier-Stokes equations from the spatial x, y, z coordinates to the curvilinear system in ξ, η, ζ coordinates. These coordinates are aligned along the mesh, with the boundaries at constant ξ, η and ζ. A new time variable $\eth$ may also be defined as part of the coordinate transformations. These transformation variables are defined mathematically as functions of time and the Cartesian variables:

$$\begin{aligned} \xi &\equiv \xi(x, y, z, t) \\ \eta &\equiv \eta(x, y, z, t) \\ \zeta &\equiv \zeta(x, y, z, t) \\ \eth &\equiv \eth(t) \end{aligned} \tag{122}$$

Firstly, the geometric conservation law must be fulfilled by this system. This requires that the change in volume of the system is equal to the flux of the cell volumes across the faces of the volume. The differential form of the geometric conservation law can be expressed as:

$$\mathscr{J}_{\eth} + (\mathscr{J}\xi_t)_\xi + (\mathscr{J}\eta_t)_\eta + (\mathscr{J}\zeta_t)_\zeta = 0 \tag{123}$$

Where subscripts indicate partial differentiation with respect to that variable. $\mathscr{J}$ is the Jacobian matrix for the transform from Cartesian to general curvilinear space coordinates :

$$\mathscr{J} = \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} \tag{124}$$

Solving this equation in addition to the transformed Navier-Stokes equations avoids spurious, grid motion induced flowfield errors [106]. The compressible Navier-Stokes equations in Cartesian coordinates can be written as:

$$\frac{\partial \boldsymbol{\mathscr{U}}}{\partial t} + \frac{\partial \boldsymbol{\mathscr{F}}}{\partial x} + \frac{\partial \boldsymbol{\mathscr{G}}}{\partial y} + \frac{\partial \boldsymbol{\mathscr{H}}}{\partial z} = \frac{\partial \boldsymbol{\mathscr{F}}_v}{\partial x} + \frac{\partial \boldsymbol{\mathscr{G}}_v}{\partial y} + \frac{\partial \boldsymbol{\mathscr{H}}_v}{\partial z} \tag{125}$$

Where $\boldsymbol{\mathscr{U}}$ is the vector of conserved quantities, $\boldsymbol{\mathscr{F}}$, $\mathbf{G}$ and $\boldsymbol{\mathscr{H}}$ are vectors of inviscid fluxes, and $\boldsymbol{\mathscr{F}}_v$, $\boldsymbol{\mathscr{G}}_v$ and $\boldsymbol{\mathscr{H}}_v$ are the corresponding viscous flux vectors. These can be defined as follows:

$$\boldsymbol{\mathscr{U}} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix} \tag{126}$$

Here ρ is the fluid density, u, v and w are the fluid velocity components in the x, y and z directions respectively and $E = e/\rho + (u^2 + v^2 + w^2)/2$ is the total energy density. e is the specific internal energy. $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$ are defined as:

$$\boldsymbol{\mathcal{F}} = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho uw \\ \rho uH \end{bmatrix}, \boldsymbol{\mathcal{G}} = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + P \\ \rho vw \\ \rho vH \end{bmatrix}, \boldsymbol{\mathcal{H}} = \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + P \\ \rho wH \end{bmatrix} \tag{127}$$

where P is the fluid pressure and $H = E + P/\rho$. The viscous flux vectors are defined as:

$$\boldsymbol{\mathcal{F}}_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ \mathscr{b}_x \end{bmatrix}, \boldsymbol{\mathcal{G}}_v = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ \mathscr{b}_y \end{bmatrix}, \boldsymbol{\mathcal{H}}_v = \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ \mathscr{b}_z \end{bmatrix} \tag{128}$$

Where τ is the viscous stress, defined using Einstein summation notation as:

$$\begin{aligned} \tau_{ij,i\neq j} &= (\mu + \mu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ \tau_{ii} &= \lambda \frac{\partial u_i}{\partial x_i} + 2(\mu + \mu_t)\frac{\partial u_i}{\partial x_i} \\ \lambda &= -\frac{2}{3}(\mu + \mu_t) \end{aligned} \tag{129}$$

where μ is the dynamic viscosity and μ_t is the dynamic turbulent viscosity (turbulence modelling is covered is Section 5.2.6). $\mathscr{b}_i$ is defined as:

$$\mathscr{b}_i = \frac{\gamma}{\gamma - 1}\left(\frac{\mu}{Pr} + \frac{\mu_t}{Pr_t}\right)\frac{\partial e}{\partial x_i} + u_j \tau_{ji} \tag{130}$$

where γ is the ratio of specific heats, Pr is the Prandtl number and Pr_t is the turbulent Prandtl number. When transformed, the Navier-Stokes equations in curvilinear coordinates are:

$$\frac{\partial \hat{\boldsymbol{\mathcal{U}}}}{\partial \eth} + \frac{\partial \hat{\boldsymbol{\mathcal{F}}}}{\partial \xi} + \frac{\partial \hat{\boldsymbol{\mathcal{G}}}}{\partial \eta} + \frac{\partial \hat{\boldsymbol{\mathcal{H}}}}{\partial \zeta} = \frac{\partial \hat{\boldsymbol{\mathcal{F}}}_v}{\partial \xi} + \frac{\partial \hat{\boldsymbol{\mathcal{G}}}_v}{\partial \eta} + \frac{\partial \hat{\boldsymbol{\mathcal{H}}}_v}{\partial \zeta} \tag{131}$$

where the variables with hats are the transformed variables, defined as:

$$\begin{aligned} \hat{\boldsymbol{\mathcal{U}}} &= \mathscr{J}\boldsymbol{\mathcal{U}} \\ \hat{\boldsymbol{\mathcal{F}}} &= \mathscr{J}(\xi_t \boldsymbol{\mathcal{U}} + \xi_x \boldsymbol{\mathcal{F}} + \xi_y \boldsymbol{\mathcal{G}} + \xi_z \boldsymbol{\mathcal{H}}) \\ \hat{\boldsymbol{\mathcal{G}}} &= \mathscr{J}(\eta_t \boldsymbol{\mathcal{U}} + \eta_x \boldsymbol{\mathcal{F}} + \eta_y \boldsymbol{\mathcal{G}} + \eta_z \boldsymbol{\mathcal{H}}) \\ \hat{\boldsymbol{\mathcal{H}}} &= \mathscr{J}(\zeta_t \boldsymbol{\mathcal{U}} + \zeta_x \boldsymbol{\mathcal{F}} + \zeta_y \boldsymbol{\mathcal{G}} + \zeta_z \boldsymbol{\mathcal{H}}) \\ \hat{\boldsymbol{\mathcal{F}}}_{\boldsymbol{v}} &= \mathscr{J}(\xi_x \boldsymbol{\mathcal{F}}_{\boldsymbol{v}} + \xi_y \boldsymbol{\mathcal{G}}_{\boldsymbol{v}} + \xi_z \boldsymbol{\mathcal{H}}_{\boldsymbol{v}}) \\ \hat{\boldsymbol{\mathcal{G}}}_{\boldsymbol{v}} &= \mathscr{J}(\eta_x \boldsymbol{\mathcal{F}}_{\boldsymbol{v}} + \eta_y \boldsymbol{\mathcal{G}}_{\boldsymbol{v}} + \eta_z \boldsymbol{\mathcal{H}}_{\boldsymbol{v}}) \\ \hat{\boldsymbol{\mathcal{H}}}_{\boldsymbol{v}} &= \mathscr{J}(\zeta_x \boldsymbol{\mathcal{F}}_{\boldsymbol{v}} + \zeta_y \boldsymbol{\mathcal{G}}_{\boldsymbol{v}} + \zeta_z \boldsymbol{\mathcal{H}}_{\boldsymbol{v}}) \end{aligned} \tag{132}$$

5.2.2 Riemann Solver

Flamenco employs the HLLC Riemann solver based on the works of Toro et al. [108] and Bagabir and Drikakis [109]. This defines the flux vector at the interface between cells $\mathbf{F}_{i+\frac{1}{2}}$:

$$\mathbf{F}_{\mathbf{i+\frac{1}{2}}} = \begin{cases} \mathbf{F_L} & \text{if } S_L > 0 \\ \mathbf{F_L^*} & \text{if } S_L \leq 0 < S_* \\ \mathbf{F_R^*} & \text{if } S_* \leq 0 \leq S_R \\ \mathbf{F_R} & \text{if } S_R < 0 \end{cases} \tag{133}$$

Where S_L and S_R are the wave speeds of the left and right acoustic waves respectively, and S_* is the speed of the contact wave. Other variables marked with * refer to values in the region between the left and right acoustic waves. The subsonic fluxes are defined as:

$$\mathbf{F_K^*} = M \begin{pmatrix} \rho_K^* S_* \\ (\rho u)_K^* S_* + p^* \check{\xi}_x \\ (\rho v)_K^* S_* + p^* \check{\xi}_y \\ (\rho w)_K^* S_* + p^* \check{\xi}_z \\ (\rho_K^* E_K^* + p^*) S_* \end{pmatrix} \tag{134}$$

where $K = L, R$ refers to the left and right hand sides of the contact surface, and $\check{\xi}_a = \xi_a / M$ where the magnitude M of the vector $\xi_{x,y,z}$ is $M = \sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}$.

5.2.3 Fluxes

The limiter of Kim and Kim [110] is used to provide 5^{th} order accuracy in regions of smooth flow. As a total variation diminishing scheme, only 1^{st} order accuracy is achieved at discontinuities. The primitive variables Γ at cell interfaces are calculated using Equation (135):

$$\begin{aligned} \Gamma_{i+\frac{1}{2}}^L &= \bar{\Gamma}_i + \frac{1}{2} \varphi(\varrho_i^L) \Delta\Gamma_{i-\frac{1}{2}} \\ \Gamma_{i+\frac{1}{2}}^R &= \bar{\Gamma}_{i+1} + \frac{1}{2} \varphi(\varrho_{i+1}^R) \Delta\Gamma_{i+\frac{3}{2}} \end{aligned} \tag{135}$$

The L and R subscripts refer to the left and right states of a cell interface. Integer subscripts refer to cell i, and the $1/2$ subscripts pertain to the cell interfaces, for example $\Gamma_{i+\frac{1}{2}}$ refers to Γ at the interface between cells i and $i+1$, and $\Gamma_{i-\frac{1}{2}}$ to that between cells i and $i-1$. $\bar{\Gamma}$ is the cell averaged primitive values, φ is the flux limiter function, and ϱ is the ratio of variation, defined as:

$$\begin{aligned} \varrho_{i+n}^L &= \frac{\Delta\Gamma_{i+n+\frac{1}{2}}}{\Delta\Gamma_{i+n-\frac{1}{2}}} \\ \varrho_{i+n}^R &= \frac{\Delta\Gamma_{i+n-\frac{1}{2}}}{\Delta\Gamma_{i+n+\frac{1}{2}}} \end{aligned} \tag{136}$$

For $n \in \mathbb{N}$. $\Delta\Gamma$ is the variation at the cell interface:

$$\Delta\Gamma_{i+n+\frac{1}{2}} = \bar{\Gamma}_{i+n+1} - \bar{\Gamma}_{i+n} \tag{137}$$

The limiter φ is expressed in Equations 138.

$$\begin{aligned} \varphi(\varrho_i^L) &= \max(0, \min(2, 2\varrho_i^L, \vartheta^L)) \\ \varphi(\varrho_{i+1}^R) &= \max(0, \min(2, 2\varrho_{i+1}^R, \vartheta^R)) \end{aligned} \tag{138}$$

Where

$$\begin{aligned} \vartheta^L &= \frac{-2/\varrho_{i-1}^L + 11 + 24\varrho_i^L - 3\varrho_i^L\varrho_{i+1}^L}{30} \\ \vartheta^R &= \frac{-2/\varrho_{i+2}^R + 11 + 24\varrho_{i+1}^R - 3r_{i+1}^R\varrho_i^R}{30} \end{aligned} \tag{139}$$

Viscous terms are evaluated to 2nd order accuracy with a standard central difference scheme. Furthermore, grid metrics are also evaluated to 2nd order accuracy.

5.2.4 Time Stepping

A 4-stage Strong Stability-Preserving Runge-Kutta method of Spiteri and Ruuth [111] achieves 2nd order accuracy in time with explicit time stepping. Firstly, the cell Jacobians are updated according to:

$$J^{(1)} = J^n + \frac{1}{3}\Delta V \tag{140}$$

$$J^{(\alpha)} = J^{(\alpha-1)} + \frac{1}{3}\Delta V, \alpha = 2, 3 \tag{141}$$

$$J^{n+1} = \frac{1}{4}J^n + \frac{3}{4}J^{(3)} + \frac{1}{4}\Delta V \tag{142}$$

where the bracketed superscripts refer to the sub-steps and the plain superscripts refer to the timesteps. ΔV is the volume flux, which is evaluated using the cell face velocities and the timestep size. After the Jacobians are updated and fluxes evaluated, the flow variables are updated as follows:

$$\mathbf{U}^{(1)} = \frac{J^n}{J^{(1)}}\mathbf{U}^n - \frac{1}{3}\frac{\Delta t}{J^{(1)}}\mathbf{F}\left(\mathbf{U}^n\right) \tag{143}$$

$$\mathbf{U}^{(\alpha)} = \frac{J^{(\alpha-1)}}{J^{(\alpha)}}\mathbf{U}^{(\alpha-1)} - \frac{1}{3}\frac{\Delta t}{J^{(\alpha)}}\mathbf{F}\left(\mathbf{U}^{(\alpha-1)}\right), \alpha = 2, 3 \tag{144}$$

$$\mathbf{U}^{n+1} = \frac{1}{4}\frac{J^n}{J^{n+1}}\mathbf{U}^n + \frac{3}{4}\mathbf{U}^{(3)} - \frac{1}{4}\frac{\Delta t}{J^{n+1}}\mathbf{F}\left(\mathbf{U}^{(3)}\right) \tag{145}$$

where Δt is the timestep size, $\mathbf{U}$ here is the vector of discretised variables and $\mathbf{F}(\mathbf{U})$ is the vector of the fluxes of $\mathbf{U}$.

5.2.5 Low Mach Correction

The low Mach correction of Thornber et al. [112] is used to improve the resolution of low Mach flow features such as the wake and blade root flow. This method modifies the velocities at cell interfaces prior to the solution of the Riemann problem. Where the L and R subscripts refer to the left and right cell interfaces, the low Mach correction can be defined as:

$$\mathbf{U}_{L,LM} = \frac{\mathbf{U}_L + \mathbf{U}_R}{2} + \varkappa \frac{\mathbf{U}_L - \mathbf{U}_R}{2} \tag{146}$$

$$\mathbf{U}_{R,LM} = \frac{\mathbf{U}_L + \mathbf{U}_R}{2} + \varkappa \frac{\mathbf{U}_R - \mathbf{U}_L}{2} \tag{147}$$

where the LM subscript references low Mach corrected variables, and $\varkappa$ is defined as $\varkappa = \min(M_{local}, 1)$ with $M_{local} = \max(M_L, M_R)$ with M being the Mach number. This method was shown to reduce the kinetic energy dissipation rate mathematically and in numerical tests [112].

5.2.6 Turbulence Modelling

A Spalart-Allmaras based detached eddy simulation (DES) turbulence model is used to model the unsteady turbulent features of the flow. This model is based on the work of Spalart and Allmaras [113, 114] and Allmaras et al. [115]. This model is grounded on the definition of a working variable $\tilde{\nu}$ with the equations:

$$\nu_t = \tilde{\nu} f_{v_1},\ f_{v_1} = \frac{\chi^3}{\chi^3 + c_{v_1}^3},\ \chi = \frac{\tilde{\nu}}{\nu} \tag{148}$$

$\tilde{\nu}$ fulfils the transport equation:

$$\frac{\partial \tilde{\nu}}{\partial t} + \mathbf{u} \cdot \nabla \tilde{\nu} = \mathscr{P} - \mathscr{D} + \mathscr{R} + \frac{1}{\sigma}\left[\nabla \cdot \left(\left(\nu + \tilde{\nu}\right) \nabla \tilde{\nu}\right) + c_{b_2} \left(\nabla \tilde{\nu}\right)^2\right] \tag{149}$$

Where $\mathbf{u}$ is the vector of fluid velocities in each of the spatial dimensions. $\mathscr{P}$, $\mathscr{D}$ and $\mathscr{R}$ are the production, wall destruction and trip terms respectively. These are defined as:

$$\mathscr{P} = c_{b_1}\left(1 - f_{t_2}\right)\tilde{S}\tilde{\nu},\ \mathscr{D} = \left(c_{w_1} f_w - \frac{c_{b_1}}{\kappa^2} f_{t_2}\right)\left[\frac{\tilde{\nu}}{d}\right]^2,\ \mathscr{R} = f_{t_1}\left(\Delta u\right)^2 \tag{150}$$

Where d is the distance to the closest wall and κ is the von Kármán constant, with its value taken as 0.41. $\tilde{S}$ is the modified vorticity, defined by:

$$\tilde{S} = \begin{cases} S + \bar{S} & \bar{S} \geq -c_{v_2} S \\ S + \frac{S\left(c_{v_2}^2 S + c_{v_3} \bar{S}\right)}{\left(c_{v_3} - 2c_{v_2}\right)S - \bar{S}} & \bar{S} < -c_{v_2} S \end{cases} \tag{151}$$

$$\bar{S} = \frac{\tilde{\nu}}{k^2 d^2} f_{v_2},\quad S = \sqrt{2\Omega_{ij}\Omega_{ij}},\quad \Omega_{ij} = \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \tag{152}$$

f_{v_2} is defined as:

$$f_{v_2} = 1 - \frac{\chi}{1 + \chi f_{v_1}} \tag{153}$$

Additionally, f_w in the destruction term is calculated according to:

$$f_w = g\left(\frac{1 + {c_{w3}}^6}{g^6 + {c_{w3}}^6}\right)^{1/6}, \quad g = r + c_{w2}\left(r^6 - r\right) \tag{154}$$

with:

$$r = \begin{cases} \min(\frac{\tilde{\nu}}{\tilde{S}k^2d^2}, r_{lim}) & \text{if } \tilde{S} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{155}$$

The function f_{t_2} is defined by:

$$f_{t_2} = c_{t_3} \exp\left(-c_{t_4}\chi^2\right) \tag{156}$$

The trip function f_{t_1} is evaluated using:

$$f_{t_1} = c_{t_1} g_t \exp\left(-c_{t_2}\frac{\omega_t}{\Delta u^2}\left[d^2 + g_t^2 d_t^2\right]\right) \tag{157}$$

where g_t is defined as per Equation (158):

$$g_t = \min\left(0.1, \frac{\Delta u}{\omega_t \Delta x}\right) \tag{158}$$

In Equations 157 and 158, d_t is the distance to the trip point, ω_t is the vorticity at the trip, Δu is the difference in velocity to the trip point, and Δx is the streamwise grid spacing at the trip. The empirical constants are $c_{b_1} = 0.1355$, $\sigma = 2/3$, $c_{b_2} = 0.622$, $c_{w_1} = c_{b_1}/\kappa^2 + (1 + c_{b_2})/\sigma$, $c_{w_2} = 0.3$, $c_{w_3} = 2$, $c_{v_1} = 7.1$, $c_{t_3} = 1.2$, $c_{t_4} = 0.5$, and $r_{lim} = 10$. These equations can also be combined with the mass conservation equation to create a conservative form:

$$\begin{gathered} \frac{\partial \rho\tilde{\nu}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\tilde{\nu}) = c_{b_1}(1 - f_{t_2})\tilde{S}\rho\tilde{\nu} + \\ \frac{1}{\sigma}\left[\nabla \cdot (\mu + \rho\tilde{\nu})\nabla\tilde{\nu} + c_{b_2}\rho(\nabla\tilde{\nu})^2 + (\mu + \rho\tilde{\nu})\nabla\rho \cdot \nabla\tilde{\nu}\right] - c_{w_1}\rho f_w\left(\frac{\tilde{\nu}}{d}\right)^2 \end{gathered} \tag{159}$$

This equation is solved alongside the Navier-Stokes equations as defined in Section 5.2.1.

No slip walls should have the boundary condition $\tilde{\nu} = 0$. The freestream values of $\tilde{\nu}$ should be chosen according to the desired boundary layer behaviour. For fully turbulent flows, a value of $\tilde{\nu}$ so that $\tilde{\nu}/\nu = 3 - 5$ is sufficient to guarantee correct behaviour.

There are several changes to the model in Equation (159) to ensure stable behaviour in regions of negative eddy viscosity. Firstly, the eddy viscosity is prevented from becoming negative:

$$\mu_t = \begin{cases} \rho\tilde{\nu} f_{v_1} & \text{if } \tilde{\nu} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{160}$$

The production and destruction terms are modified for $\tilde{\nu} \leq 0$:

$$\mathscr{P} = \begin{cases} c_{b_1}(1 - f_{t_2})\tilde{S}\rho\tilde{\nu} & \text{if } \tilde{\nu} > 0 \\ c_{b_1}(1 - c_{t_3})S\rho\tilde{\nu} & \text{otherwise} \end{cases} \tag{161}$$

$$\mathscr{D} = \begin{cases} \rho\left(c_{w_1} f_w - \frac{c_{b_1}}{\kappa^2} f_{t_2}\right)\left(\frac{\tilde{\nu}}{d}\right)^2 & \text{if } \tilde{\nu} > 0 \\ -c_{w_1}\rho\left(\frac{\tilde{\nu}}{d}\right)^2 & \text{otherwise} \end{cases} \tag{162}$$

Finally, the diffusion term is also modified:

$$\text{diffusion} = \begin{cases} \frac{1}{\sigma}\nabla\cdot(\mu + \rho\tilde{\nu})\nabla\tilde{\nu}, & \text{if } \tilde{\nu} > 0 \\ \frac{1}{\sigma}\nabla\cdot(\mu + \rho\tilde{\nu} f_n)\nabla\tilde{\nu}, & \text{otherwise} \end{cases} \tag{163}$$

$$f_n = \frac{c_{n_1} + \chi^3}{c_{n_1} - \chi^3} \tag{164}$$

Here the value of c_{n_1} is 16.

Using a Spalart-Allmaras DES method places the turbulence modelling in a similar model fidelity to a number of blade-resolved studies mentioned in this thesis. The work of Fitzgibbon et al. employed a fully turbulent Wilcox $k-\omega$ shear-stress transport (SST) model [15, 116], as did Barakos and Garcia [117]. Jain used the Menter SST $k-\omega$ model in the FUN3D solver, while the OVERFLOW solver used the same model, but with a SST based DES for the wake [118]. Lim et al. used a Menter $k-\omega$ SST Delayed Detached Eddy Simulation (DDES) for one rotor, and a Spalart-Allmaras DDES for the other [17]. Bodling and Potsdam used Spalart-Allmaras DES and DDES turbulence modelling in the various solvers for their simulations [119].

5.2.7 Viscosity Modelling

Flamenco includes the option to model the dynamic viscosity of a gas using Sutherland's Law [120]. Sutherland's Law can be expressed as:

$$\mu = \mu_0 \left(\frac{\mathscr{T}}{\mathscr{T}_0}\right)^{\frac{3}{2}} \frac{\mathscr{T}_0 + S}{\mathscr{T} + S} \tag{165}$$

where S is the Sutherland constant, $\mathscr{T}_0$ is the reference temperature and μ_0 is the dynamic viscosity at the reference temperature. This is valid for gases above their critical temperatures and that do not significantly deviate from Boyle's Law. This model was used in the forward flight simulations in this thesis to improve accuracy given the variation of temperature in the flowfield. The constants were taken to have values of $S = 110.4K$, $\mathscr{T}_0 = 273.15K$ and $\mu_0 = 1.716 \cdot 10^{-5}\frac{kg}{m\cdot s}$. These values are

compared to tabulated data of the dynamic viscosity of air at atmospheric pressure in Figure 36. This tabulated data is taken from Pritchard [121] and Incropera et al. [122]. The temperature range 120K - 380K covers the approximate range seen in the simulations. Within this range,the maximum error between the Sutherland's approximation and the data from Incropera et al. [122] is 0.525% and the mean error is 0.279%. Similarly, the maximum error to the data from Pritchard [121] is 0.288% and the mean error is 0.130%.

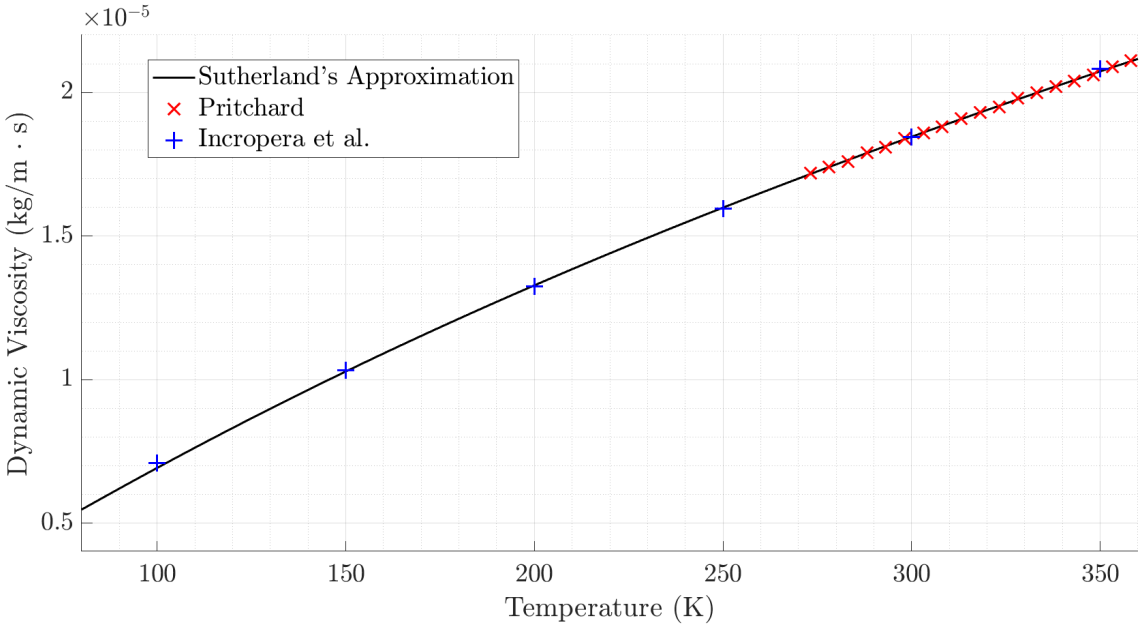


Figure 36: Comparison of dynamic viscosity calculated using Sutherland's law to measurement data

5.2.8 Wall Function

Flamenco employs a wall function of a form similar to the one function model proposed by Spalding [123]. Flamenco iteratively solves a law of the wall of the form:

$$y^+ = u^+ + e^{-\kappa B}\left(e^{\kappa u^+} - 1 - \kappa u^+ - \frac{1}{2}\left(\kappa u^+\right)^2 - \frac{1}{6}\left(\kappa u^+\right)^3\right) \tag{166}$$

with κ being the von Kármán constant, and $B = 5.5$. The wall variables y^+ and u^+ are defined as:

$$y^+ = \frac{y u_\tau}{\nu}, u^+ = \frac{u}{u_\tau}, u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{167}$$

where y is the distance to the wall, u is the local flow speed and τ_w is the wall shear stress. Solving Equation (166) gives u_τ, which is then used to modify the local turbulent viscosity:

$$\mu_t = \max\left(0, \rho\left(\frac{u_\tau^2}{\frac{\partial u}{\partial y}} - \nu\right)\right) \tag{168}$$

The velocity gradient at the wall may be approximated as u/y given the no-slip condition.

6 Simulations of Isolated Rotors in Hover

This section covers the setup and results for the simulations of isolated rotors in hover. The operating conditions will first be described, followed by the details of the mesh generation. After this, an analysis of the results is presented, including examinations of the integrated loads, sectional loads and flowfield. This chapter includes material presented at the 49$^{\text{th}}$ European Rotorcraft Forum [124].

6.1 Simulation Description

The following atmospheric conditions were used for the simulations of the isolated rotors in hover:

- Velocity $\mathbf{u} = (0, 0, 0)\, m/s$
- Pressure $P = 87510.5Pa$
- Density $\rho = 0.9893 kg/m^3$
- Viscosity $\mu = 1.88 \cdot 10^{-5} kg/ms$

These ambient conditions were chosen to match the work of Widjaja et al. [5] in order to enable a more efficient and direct comparison of results. ASM simulations were run at collective angles of 2°, 4°, 6°, 8°, 10°, 11° and 12°, again to allow direct comparison with the work of Widjaja et al. Due to computational constraints, the blade resolved simulations were only performed at 4°, 8°, 10° and 12°. This still covers the majority of the collective pitch range, spanning 8° of the 10° collective range.

6.2 Mesh Generation

6.2.1 ASM

The meshes for the hover simulations were cuboid in shape, with boundaries $3.51R$ above and to the sides of the rotor, and $7.03R$ below the rotor. The mesh consists of four sections: the rotor region, near-wake region, the far-wake region and background mesh. The rotor and wake regions are cylindrical regions of smaller cell size, with sizes as given in Table 13.

Table 13: Sizes of the refinement regions of the ASM hover simulation mesh

Mesh Region	$\mathbf{z_{max}/R}$	$\mathbf{z_{min}/R}$	**Radius/R**
Rotor	0.1054	-0.1054	1.025
Near-wake	0.1405	-0.7028	1.025
Far-wake	0.3513	-3.514	1.025

The edge lengths of the cells in the rotor region were $0.091c$. The cell edge length in the near-wake region was doubled to $0.18c$, and quadrupled in the far-wake region to $0.37c$. The background mesh had an edge length 32 times larger than in the rotor region. This meshing strategy produced meshes of 4 102 893 cells. Figure 37 shows a slice through the ASM hover mesh. This figure shows the three refinement zones clearly as well as the relative refinement of these zones.

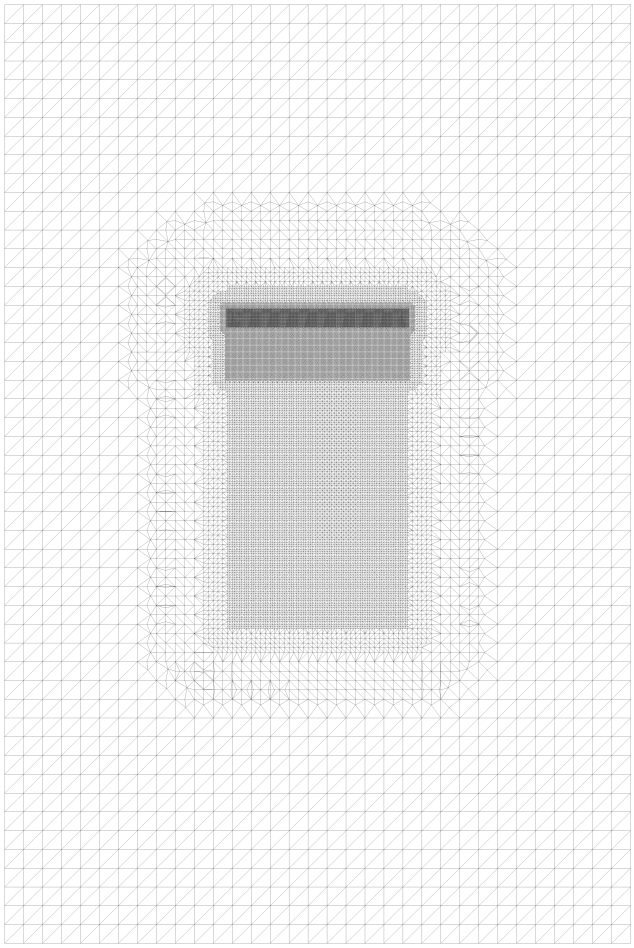


Figure 37: Slice through the ASM hover mesh

With no representation of the hub, there is only one boundary to the mesh since there is no explicit modelling of the blade surfaces when using the ASM. As a The OpenFoam boundary conditions used for each variable can be seen in Table 14.

The *totalPressure* BC enforces a constant total pressure across the boundary. The *pressureInletOutletParSlipVelocity* boundary condition acts as a zero-gradient condition when there is flow outwards across the boundaries, and calculates the velocity from the flux and specified pressure when there is an inflow. A slip condition is applied in the direction tangential to the boundary. *inletOutlet* acts as a zero-gradient boundary condition for outflow, and is set to a fixed value where there is flow into the domain.

Table 14: OpenFOAM Boundary conditions for ASM isolated hover simulations

Variable	**Boundary Condition**
P	*totalPressure*
U	*pressureInletOutletParSlipVelocity*
$\tilde{\nu}$	*inletOutlet*

6.2.2 Blade Resolved Simulation

The blade resolved simulations use a structured mesh of 18 412 207 cells with a target $y^+ = 100$ at the blade tip, estimated with the freestream conditions, blade tip chord length and flat-plate boundary layer theory. Given that the simulation is of a rotor in hover, there are no yaw, pitch or flap variations during the rotation of the blades, and therefore the geometry and flowfield should be rotationally symmetric. As the ARC helicopter rotor is a four bladed rotor, the simulation domain was reduced to a 90° section comprising of one blade and a quarter of the rotor hub. Periodic boundary conditions allow the flowfield to be simulated as if the full cylindrical domain were used.

The mesh can be thought of as having a number of key sections: the near blade mesh, the near-wake mesh and the far-wake mesh. The near blade mesh is an O-grid around the blade which aims to provide high resolution of the boundary layer. The wake meshing is based on the work of Lim et al. [17], with the near-wake mesh spanning total volume from $0.2R$ above the blade to $0.4R$ below the blade and the far-wake stretching from $0.5R$ above the blade to $-2.5R$ below, excluding the aforementioned areas. The near-wake in Lim et al. was refined to an edge length of $0.05c$ to $0.09c$ [17], which agrees well with $0.05c_{tip}$ from Bodling and Potsdam [119], a study focused on specifically resolving secondary vortex structures in rotor wake. This level of mesh refinement was impractical to simulate with the available computational resources. Instead, the near-wake cells had an edge length of $0.127c$ in the vertical direction, and the edge length in the angular direction ranged from $0.00261c$ under the trailing edge of the blade to $0.225c$ ahead and behind the blade. The radial edge lengths varied greatly from $0.000476c$ just below the blade tip to $0.318c$ below the centre of the blade. The far-wake cells have edge length $0.324c$ in the vertical direction and $0.225c$ in the radial direction. The angular edge lengths were similar to the near-wake region. Figure 38 shows the blade tip surface mesh, as well as a slice through the near blade and near-wake mesh. Nonparallel lines though the mesh are artefacts of the visualisation software. The density of the near blade O-grid is visible, as well as the clustering of points near the blade tip. The near-wake also provides the highest mesh density in the areas closest to the blade.

The mesh extends 10R above the plane of the rotor, 20R below the plane of the rotor and 9.77R radially to minimise the effect of the boundary conditions on the flow near the rotor blade. These boundary distances are comparable to Yüksel's work [125] and exceed all other studies with similar cylinder section domains. For

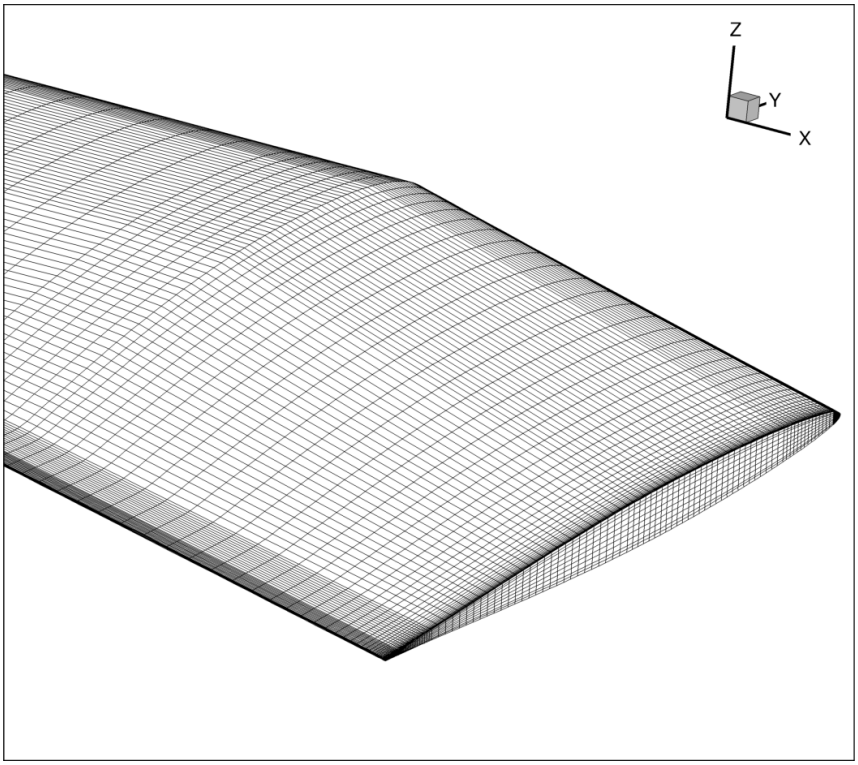


Figure 38: Hover mesh - blade surface mesh and a slice through the near blade and near-wake mesh

this reason, farfield style boundary conditions are acceptable on these faces. The domain size information be seen in Table 15, which lists a number of published mesh sizes for aerodynamic simulations of helicopter rotors. The work of Narducci and Tadghighi does not explicitly state the shape of the grid, but as the background mesh used Cartesian cells, it can be assumed that this domain was cuboid [11]. Jain [126] and Narducci and Tadghighi [11] both employ adaptive mesh refinement in the background meshes, which may help simulate such a large domain without having excessive computational costs.

There are four boundary conditions used for the blade resolved Flamenco simulations. The blade was modelled as a no-slip wall. The hub was modelled as an

Table 15: Comparison of hover mesh parameters

Author	**Radial Dist.**	**Top Dist.**	**Bottom Dist.**
Barakos, Garica [117]	6R	3R	6R
Garipova, Batrakov [127]	3.2R	2R	3.2R
Johnson [128]	> 5R	> 5R ($64c$)	> 5R ($50c$)
Potsdam, Strawn [129]	5R	5R	5R
Yüksel [125]	10R	10R	15R
Narducci and Tadghighi	20R (to sides - cuboid)	10R	30R
Jain [126]	20R in all directions - cuboid		

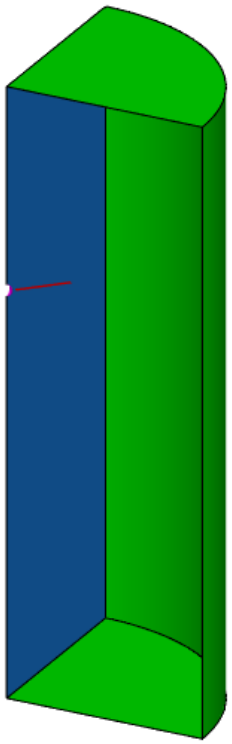


Figure 39: Diagram of boundary conditions for the blade resolved hover mesh, without rear periodic boundary (Green - inlet-outlet, blue - periodic, red - no slip wall, magenta - inviscid wall)

inviscid wall to reduce the mesh refinement requirements in a geometrically complex region. Despite being unphysical, this was seen as an acceptable compromise as force measurements weren't taken for the rotor hub, and hence the exact details of the boundary layer are insignificant. The boundaries ahead and behind of the blade are linked periodic boundary conditions. These boundary conditions work by setting the n^{th} layer ghost cell values to the corresponding flow values n cells in from the other periodic boundary, and then rotating the velocities by 90° about the vertical axis in the correct direction. The top, radial and bottom boundaries use an inlet-outlet boundary condition. This condition extrapolates the boundary values for supersonic outflow, and uses isentropic flow relations to compute both the subsonic outflow and inflow conditions, using specified ambient/total flow conditions. Figure 39 illustrates the domain and boundary condition groupings. The green boundaries are those with the inlet-outlet condition, the red is the no slip wall for the rotor blade and the magenta is the inviscid wall for the hub. The diagram shows the periodic boundary ahead of the blade in blue, but the rear boundary has been left out for illustrative purposes. It would close the quarter cylinder shape.

6.3 Collective Pitch Adjustment

Performing simulations at various collective angles without chimera grids, immersed boundary methods or mesh deformation methods requires the generation of a separate mesh for each angle that should be simulated. This meshing is often an intricate

and lengthy process, even when altering a complete mesh with a specified collective angle. However, with a mesh deformation method, one can instead change the collective angle from a base mesh for each simulation, saving a large amount of time and effort. However, these methods must not excessively degrade the quality of the mesh, especially in critical areas such as the boundary layer mesh and areas capturing important wake phenomena. With a base mesh of 8° collective angle, the multiscale RBF method was used to generate meshes for hover simulations at 4°, 10° and 12° collective. This was achieved by rotating the blade surface about the blade's pitch axis by the difference in desired collective angle to 8°. This subsection analyses grid metrics before and after deformation to quantitatively analyse the deformation. The following parts of this chapter cover the results of the simulations, displaying the success of this method in allowing accurate flow simulation.

The measure of grid quality for this analysis is the mesh orthogonality. Following Rendall and Allen [63], the orthogonality is calculated using a method based on the definitions provided in the work of Siebert and Dulikravich [130]. The orthogonality at a node is calculated as the average of the orthogonality at that node in the planes of each curvilinear coordinate:

$$q = 1 - \frac{q_i + q_j + q_k}{3} \tag{169}$$

Although orthogonality measures usually take a value of 0 to be perfectly orthogonal, corresponding with $\cos(90°)$, defining perfect orthogonality to be nonzero allows normalisation by the initial values [63]. Thus it is important to note that values of q_i, q_j or q_k closer to zero are more orthogonal, but values of q closer to 1 are more orthogonal. The orthogonality was calculated at each node in each mesh block, and then averaged over each block for nodes on shared block faces, edges or vertices. This maintains the same form of the local orthogonality equations for boundary nodes where one, two or four in-plane quadrilaterals meet, but also allows for orthogonality calculations at nodes that are vertices of greater numbers of in-plane quadrilaterals. The global mean orthogonality Q can also be calculated by finding the mean of all q values [63]. Correspondingly, a value of 1 for Q means that all cells are perfectly orthogonal.

As the RBF radii are small with regards to the dimensions of the mesh, the grid quality analysis will focus on the region of the mesh close to the blade. Outside the RBF radii, there is no movement in the mesh, and hence there is no change in the grid quality in these regions. Additionally, the most significant flow behaviour is that occurring close to the blade. In light of these factors, the grid quality was analysed within a box with limits defined by:

- spanwise coordinates within one chord-length of the blade root and tip
- vertical coordinates within a chord length of the blade's maximum and minimum vertical coordinates
- coordinates within a chord-length of the leading and trailing edges in the direction perpendicular to the spanwise and vertical axes

Table 16: Mesh quality parameters within a distance of c from the blade for hover meshes

Collective angle	$q_{i_{\max}}$	$q_{j_{\max}}$	$q_{k_{\max}}$	$q_{\min}$	$q_{\max}$	Q
4°	0.8171	0.8752	0.8102	0.4607	1.000	0.9713
8°	0.8188	0.8752	0.8072	0.4612	1.000	0.9709
10°	0.8198	0.8752	0.8059	0.4613	1.000	0.9705
12°	0.8209	0.8752	0.8047	0.4613	1.000	0.9701

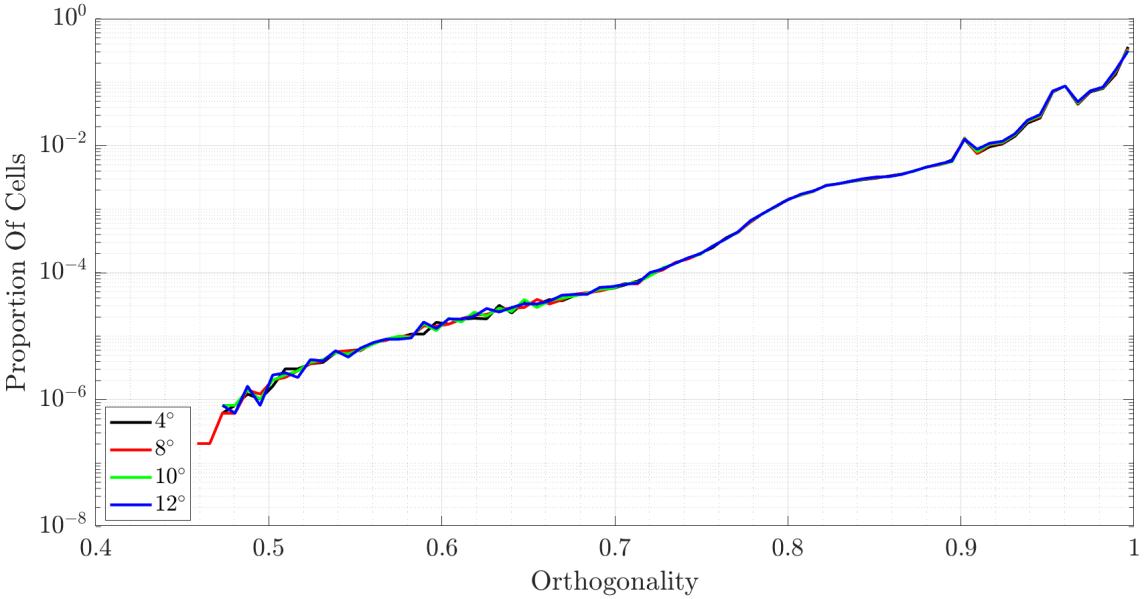


Figure 40: Distribution of cell orthogonality

A summary of the orthogonality metrics within this region is presented in Table 16.

This data shows that the quality of the grid in this region is barely affected by the mesh deformation, with differences to the undeformed mesh at 8° collective being less than 0.5% for all values. To analyse this in more detail, Figure 40 shows the distribution of cell orthogonality within this near-blade region.

It can be seen that the proportion of cells with each orthogonality value is extremely similar between each mesh, with no significant differences visible on this graph. This can also be confirmed by examining the relative change in orthogonality $\Delta q/q$ between the base mesh and deformed meshes. Figure 41 to Figure 43 display the relative orthogonality change in spanwise planes at $r \approx 0.94$. The colourbars are constructed to have a consistent range across all three figures. It is notable that the change in orthogonality is less than 1.5% across the three different collective angles. Furthermore, the largest areas of reduced grid quality generally tend to be further from blade surface.

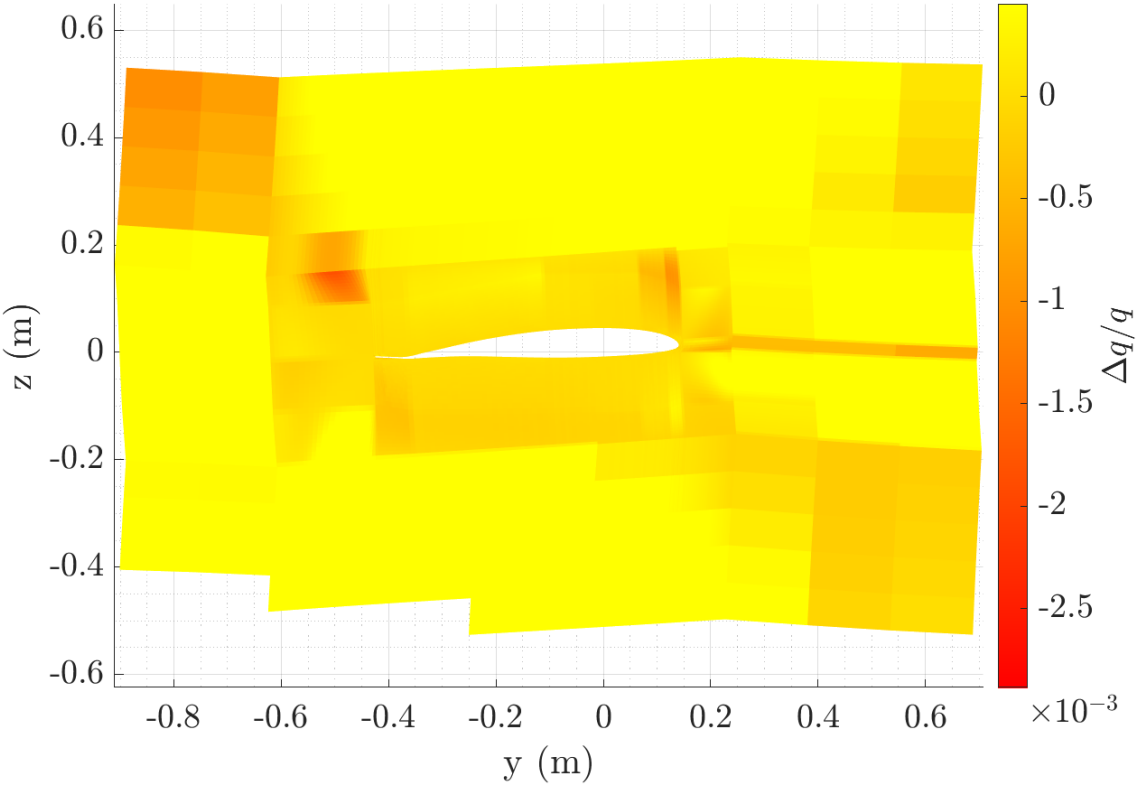


Figure 41: $\Delta q/q$ between undeformed 8° collective mesh and deformed 4° collective mesh at $x \approx 0.94R$

6.4 Integrated Loads

Figure 44 compares thrust and torque coefficient predictions in hover across the experimental results and various simulation tools as stated in Section 2.3. These values were averaged over half a rotation after reaching a periodic steady state, with half a rotation being two periods of the aerodynamic behaviour (due to the rotor being four-bladed, the behaviour is quarter-rotation periodic). A curve of best fit of the form $C_Q = aC_T^{3/2} + b$, where a and b are constants, was fitted to the Whirlstand data using MATLAB's fit function, solving for a least squares fit. This function of C_T is supported by simplified mathematical analyses of rotors in hover. According to Seddon [8] and to Johnson [9], actuator disk theory gives a relation of $C_Q \propto C_T^{3/2}$, and a simple blade element theory analysis gives the equation:

$$C_Q = \frac{\kappa C_T^{3/2}}{\sqrt{2}} + \frac{\sigma C_{do}}{8} \tag{170}$$

Where κ is an empirical factor and C_{do} is an assumed constant drag coefficient.

This fit was used to estimate the difference between the simulations and the experimental results, by calculating the difference in C_Q values at the C_T values predicted by these tools. As the collective angle values are not present for the Whirlstand data, the differences in both C_T and C_Q cannot be found. The differences in the C_Q values can be seen in Table 17.

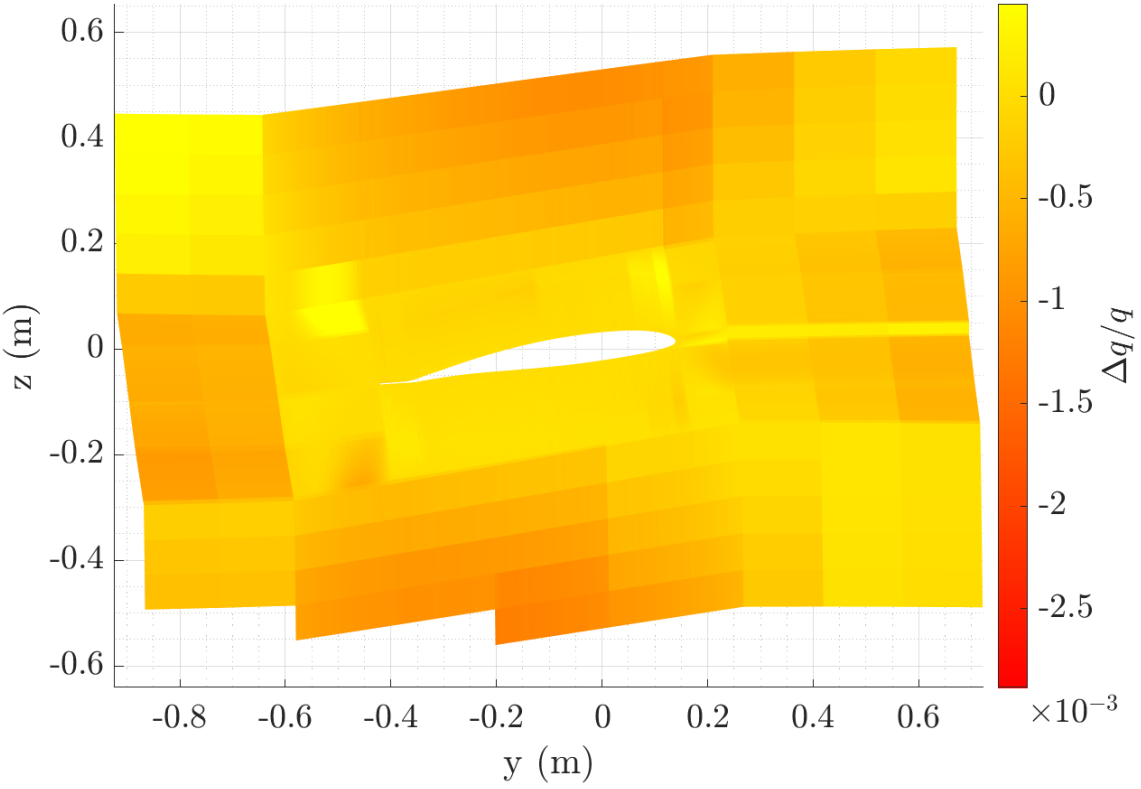


Figure 42: $\Delta q/q$ between undeformed 8° collective mesh and deformed 10° collective mesh at $x \approx 0.94R$

It should be noted that the Whirlstand data has a maximum thrust coefficient of $C_T = 8.09 \cdot 10^{-3}$, which corresponds to a collective angle of approximately 10°. This means that the errors to the Whirlstand trend above this collective angle are in essence extrapolated.

Across the flow solvers there is a general trend of higher error at low collective angles and at the upper extreme of the collective range. Helios, however, increases in accuracy with collective angle. Flamenco demonstrates an accuracy comparable to the other blade resolved solvers despite the use of a wall function. Flamenco generally overpredicted the torque values relative to the Whirlstand as both the results at 4°and 12°collective are too high, although the underprediction at 8°does not match this trend.

The ASM results are comparably accurate to the blade resolved solvers from 6°to 10°. Below this range the error is much larger, over three times the average blade-resolved error at 4°, and 1.5 times the error of Helios at 2°. Notably, the C_Q values are underestimated instead of overestimated in this range. The ASM is also less accurate above 10°, with error multiple times greater than the averaged absolute error of the blade-resolved solvers.

Figure 45 shows the FoM values plotted against C_T scaled by the rotor solidity.

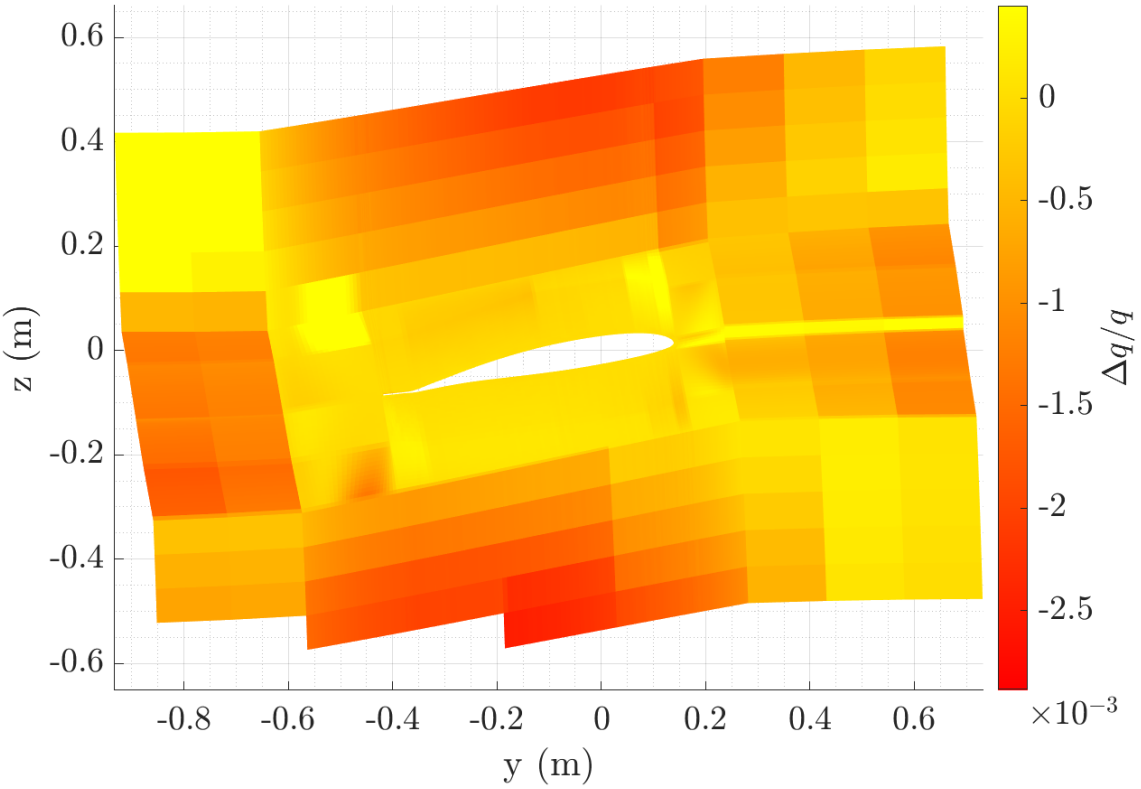


Figure 43: $\Delta q/q$ between undeformed 8° collective mesh and deformed 12° collective mesh at $x \approx 0.94R$

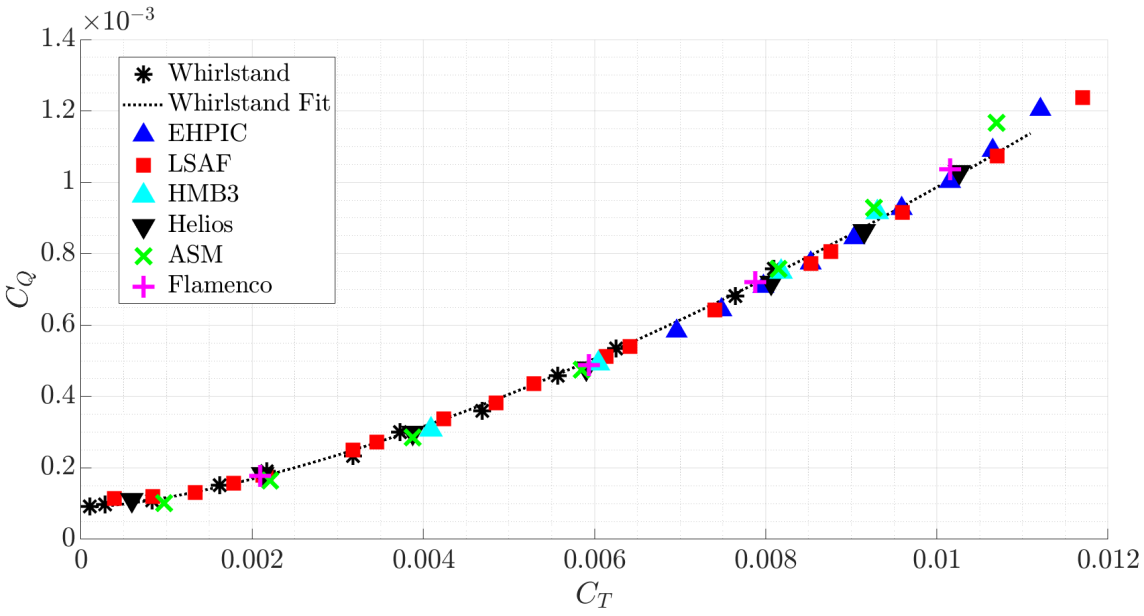


Figure 44: Comparison of integrated load predictions

Table 17: Percentage difference in C_Q between flow solvers and Whirlstand fit values

Collective (°)	**Flamenco**	**ASM**	**Helios**	**HMB3**
2	-	-12.5	8.36	-
4	1.62	-9.49	3.56	-
6	-	-6.4	-2.17	-5.05
8	-2.09	-3.1	-3.48	-3.57
10	0.588	1.18	-2.68	-0.49
11	-	4.41	-1.24	2.43
12	2.92	7.85	0.692	-

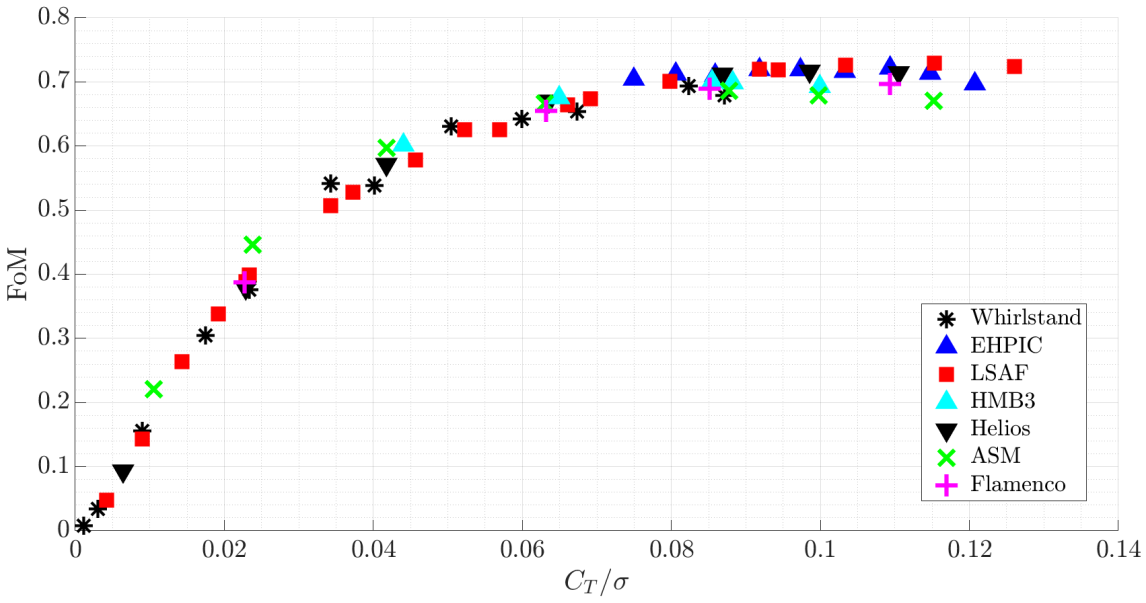


Figure 45: Comparison of figure of merit predictions

Table 18: Percentage difference in FoM between flow solvers and Whirlstand fit values

Collective	**Flamenco**	**ASM**	**Helios**	**HMB3**
2	-	18.7	-7.72	-
4	-1.59	10.5	-3.43	-
6	-	6.84	2.22	5.32
8	2.14	3.2	3.61	3.70
10	-0.583	-1.17	2.75	0.492
11	-	-4.22	1.26	-2.38
12	-2.83	-7.27	-0.687	-

Considering the FoM is computed from the thrust and torque coefficients, it doesn't strictly present any new data. However, as the FoM is a measure of efficiency, it is an important measure in rotorcraft design and therefore warrants presentation. The Whirlstand fit curve for the FoM was calculated from the corresponding values of the thrust-torque curve fit defined previously. The percentage errors between this fit and the solver predictions at the predicted C_T/σ values are listed in Table 18.

The errors to the Whirlstand FoM fit are very similar to the errors to the Whirlstand C_Q fit, with the ratio between the errors being approximately -1 ± 0.1. The exception to this is the ASM at 2° collective, with the FoM error being almost -1.5 times the C_Q error.

One reason for the increased errors of the solvers at high collective angles may be the increased difficulty of simulating the flow separation which is present at these angles. Evidence of flow separation can be seen in Figures 46 and 47. These display a cross-section of the flow at $0.95R$, with the contour displaying the momentum density in the y direction in the frame of reference of the blade. It can be seen in both figures that there is flow reversal at the rear of the upper surface, revealing the presence of a recirculation zone. This is also evidenced by the plotting of streamlines of in-plane velocity.

Although flow separation is generally difficult to simulate, the presence of flow separation is especially relevant to the ASM and Flamenco calculations. As the 2D lift and drag tables for the ASM were generated using XFOIL, these may suffer from inaccuracies as the aerofoil approaches and goes beyond stall [131]. It is also known that XFOIL is not equipped for viscous simulations of aerofoils with extreme flow separation [132]. As the Flamenco simulations employ a wall function, these too will be affected by the flow separation. The empirical correlations used to develop the Spalding wall function are likely inaccurate above the viscous sublayer in flows with pressure gradients [133], and hence the boundary layers in this part of the mesh may not be correct. As an example of this in praxis, Mockett et al. showed reduced accuracy in skin friction, separation location and reattachment location when simulating separating flow over a periodic hill when using DES with a wall function [134].

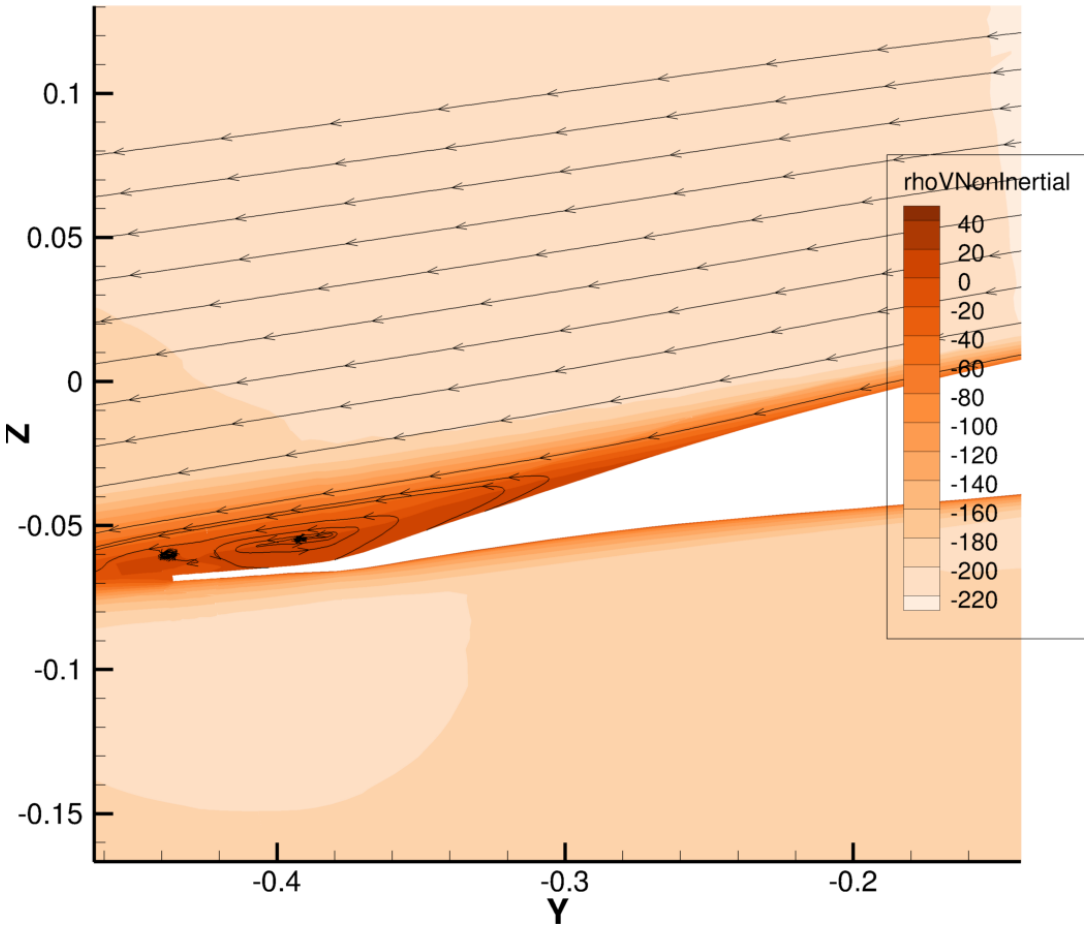


Figure 46: Flow separation at 10° collective at $0.95R$

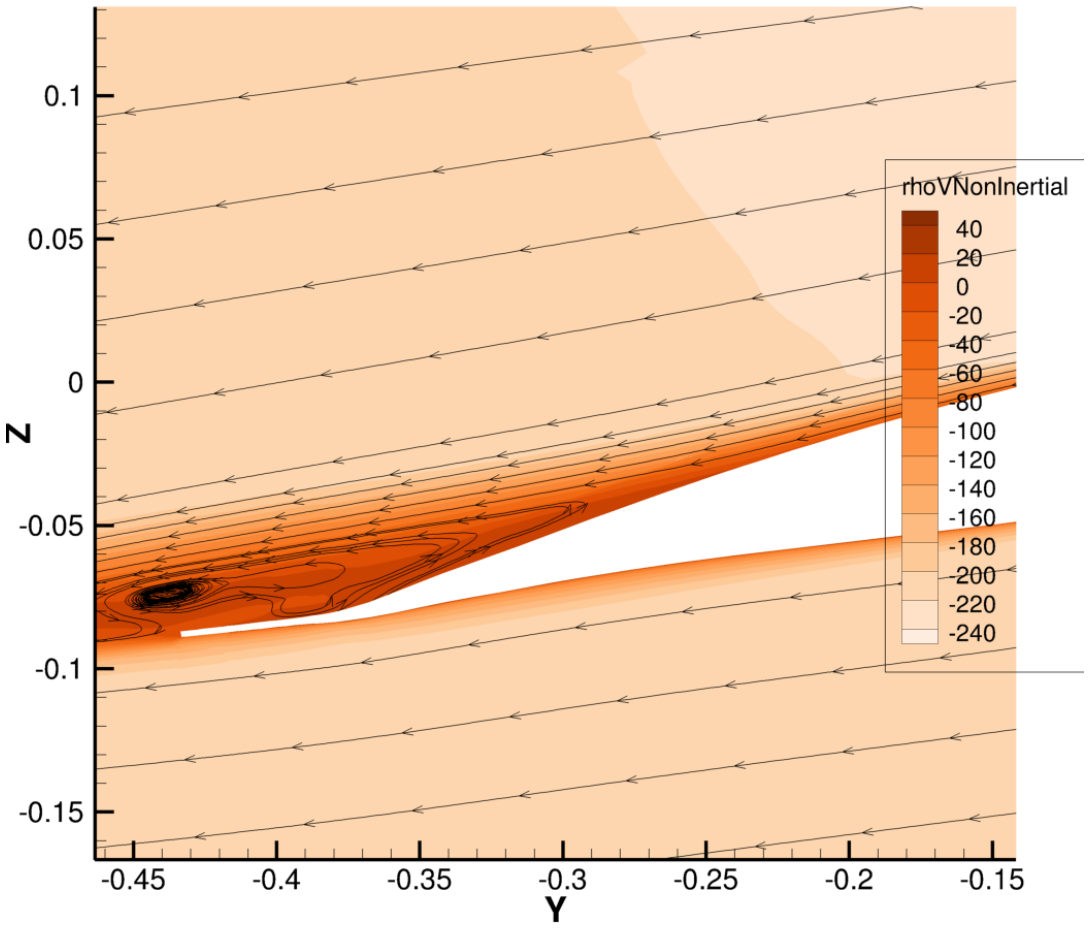


Figure 47: Flow separation at 12° collective at $0.95R$

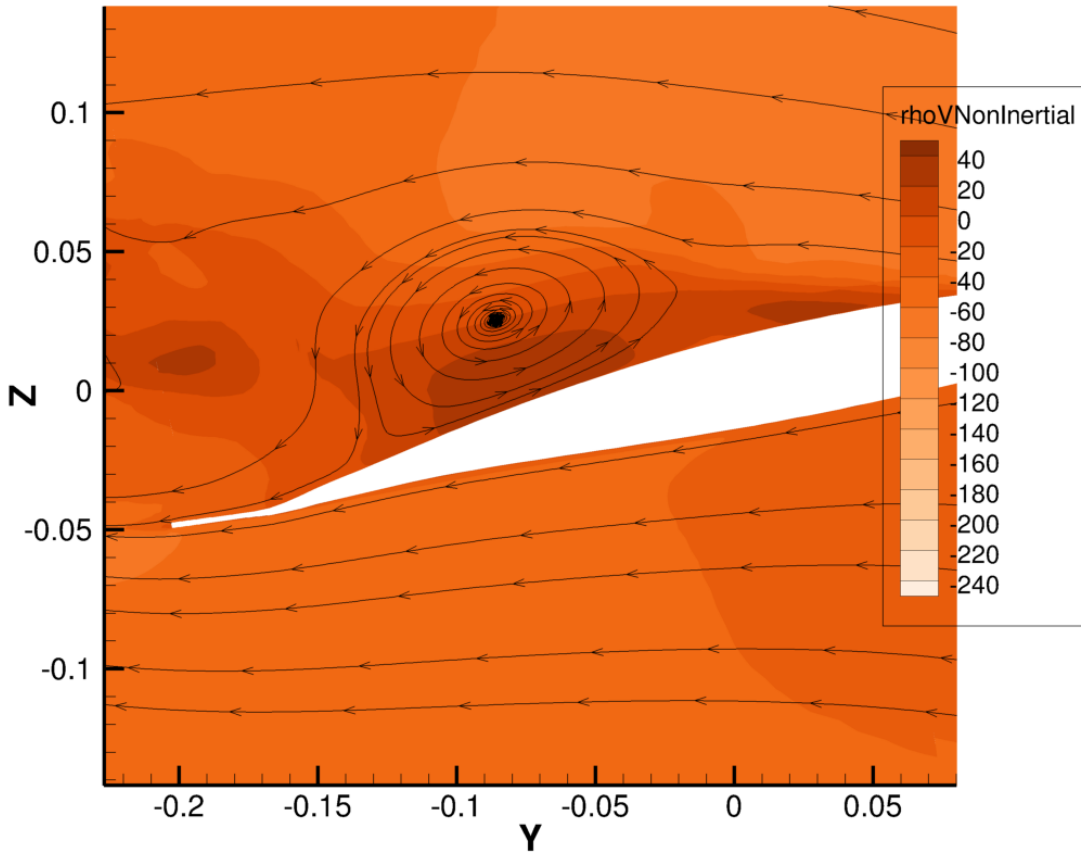


Figure 48: Flow separation at 8° collective at $0.2R$

While the issues of flow separation are most prominent at these high collective angles, there are similar issues at the blade root at lower collective angles. According to McCroskey et al. [135], at a Mach number of 0.3 the HH-02 aerofoil stalls at an angle of attack of approximately 12-13°. At 8° collective, such high local angles of attack are seen near the blade root, from $0.3R$ inwards, and the root angle of attack is approximately 11.5° at 6° collective. Although the local Mach numbers are lower than 0.3 in these areas, there is evidence of flow separation when examining the flowfields. Figure 48 shows the flowfield at 8° collective at $0.2R$, displaying large flow separation as an example. Considering this evidence for large amounts of flow separation over the blade at the high collective angles, the increased inaccuracy of the Flamenco and ASM load predictions may be accounted for by the difficulties of XFOIL and the Spalding wall function under such flow conditions.

An explanation for the larger inaccuracies at the lower extremities of the collective range may lie in the interaction between the blades and the wake features that they shed. This is most prominent at the blade root and tip, where there are vortices being shed, which the following blade passes through or very close to. This phenomenon is known as blade vortex interaction, and is widely acknowledged as an important feature of rotor-wake interaction. Blade vortex interaction can affect the aerodynamics of the rotor by changing the local angle of attack where the impingement occurs. At higher collective angles, the downwash generated by the

blades convects the tip vortices downwards, and hence reduces the impact of the vortices on the following blade. However, at low collective angles, the downwash is not sufficient to move the vortices far enough away as to have a negligible effect on the blade aerodynamics. Figure 49 displays a tip vortex and its effect in the Flamenco calculation of 4° collective.

Figure 49 shows two contour plots of the flowfield on a slice parallel to the blade axis, at a distance $c/20$ ahead of the leading edge. Both plots include mesh block edges for orientation. The leading edge of the blade lies between the two blue horizontal lines, and the blade tip is aligned with the rightmost vertical line. The blade extends in the negative x direction. The top plot displays the y component of the flow's vorticity. This can be used to identify the location of the blade tip vortex from the previous blade by finding the local peak of negative vorticity. With the in-plane anticlockwise rotation induced by this vortex in mind, the lower plot of the z direction momentum density can be examined. Looking along blade axis, it can be seen that the downwash is greater to the left (inboard) of the vortex centre. At the x location of the vortex's centre, the contour lines are angled as the induced z-velocity changes across the vortex. To the right (outboard) of the vortex location, there is even a small upwards flow near the tip. These observations suggest that the vortex changes the local angle of attack across this section of the blade, and will therefore affect the lift and torque created by this section. There should also be some secondary effects generated by the increased inboard flow generated by a tip vortex passing just below the blade, but as this flow is parallel to the blade axis one would expect the impact to be minor.

With the ability of blade tip vortices to affect the blade aerodynamics demonstrated, their simulation should now be examined. The rotor wake structures have been observed to be extremely complex in experimental studies. For example, Wolf et al. observed numerous phenomena in a particle image velocimetry study of a rotor in ground effect, which included blade shear layers, tip vortices, vortex pairing and merging, as well as shear layer deformation [136]. Such features have proven difficult to resolve fully and are sensitive to the flow solver specifics. As an example, Bodling et al. resolved these features, but required a blade mesh with 8.9 million points per blade, as well as a background mesh with edge lengths of $0.05c$, and a timestep size corresponding to 0.25° of rotation [119]. Furthermore, the solvers used 5^{th} order schemes for inviscid fluxes, 2^{nd} order schemes for time, and 2^{nd} and 4^{th} order viscous flux schemes for the near-blade solver and background flow solver respectively. Resolving the rotor wake in such high detail would have diminishing returns in accuracy, but demonstrates how computationally intensive it is to capture the flowfield in high detail. In relation to blade tip vortices more specifically, Bodling et al. also specifically investigated the convection of a isolated ring vortex as a proxy for blade tip vortex convection. Tracking the peak vortex strength showed strong sensitivity to the residual drop in both the near-blade and background solvers individually as well as to the timestep size. In particular, changes in the vortex strength preservation were seen even down to timesteps of 0.0625° [119]. Whilst the temporal and spatial discretisations of Flamenco are of a similar order to those of Bodling et al,

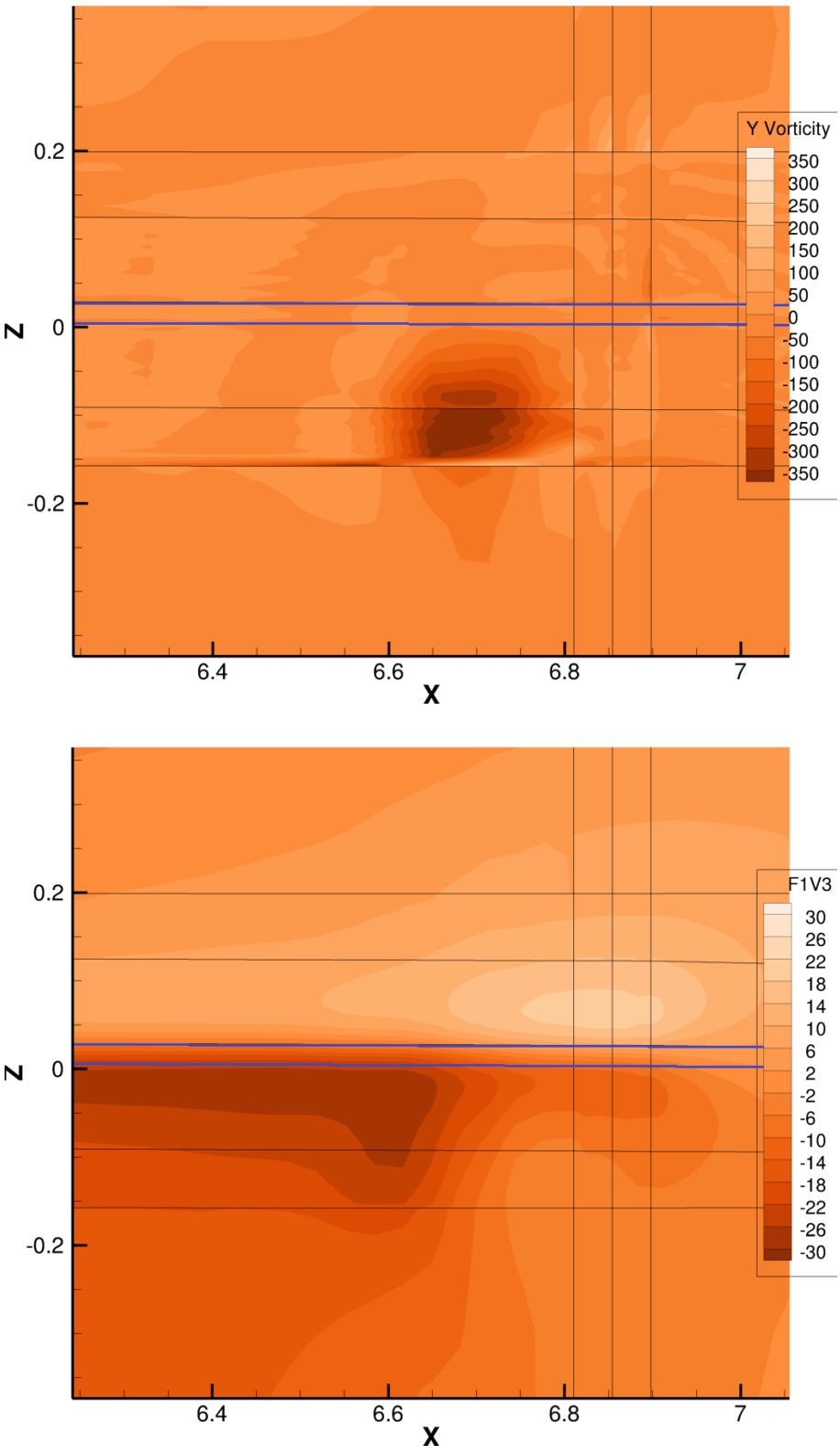


Figure 49: Y-vorticity ($/s$) (top) and Z momentum density (bottom) at 4° collective (kg/m^2s)

the cell edge length and timestep size were larger. Hence, it should be expected that these simulations would have some lack of resolution at low collective angles, where these complex wake interactions are more prominent, which could partially explain the larger differences to the Whirlstand data at low collective angles. The ASM simulations would suffer this problem to an even greater extent, with lower order schemes and somewhat less refined mesh.

Additionally, there is evidence for an underprediction of skin friction when performing boundary layer resolved simulations with many DES methods [137, 138]. This is attributed to the log-layer mismatch, an increase in the scaled boundary layer velocity above the RANS to LES transition. This is caused by the lack of resolved shear stress in this transition region. This in turn arises from the inability to resolve the appropriate eddies in this region, due to the time and length scales of the RANS solution. However, flows that amplify the instabilities in this region, such as detached flows and flows under adverse pressure gradients, may not suffer from the log-layer mismatch [138]. It is difficult to appraise if the log-layer mismatch had much of an effect on the Flamenco simulations, due to the inability to clearly separate the numerous potential sources of error.

Evaluating the integrated loads overall, it can be seen that the Flamenco simulations produced force predictions comparable to the Helios and HMB3 solvers. This lends support to the deformed mesh quality being adequate for high-accuracy simulations, as well as demonstrating that wall functions may be used in such studies. The ASM proved itself to be similarly accurate to the blade resolved solvers between collective angles of 6° and 10°. Whilst the errors grew outside of this range, these errors were still approximately 10% or lower, except at 2° collective, where accuracy was generally low for other solvers as well.

6.5 Sectional Loads

Figure 50 provides a comparison of the sectional thrust values in hover between the blade resolved solvers and ASM. Between $0.5R$ and $0.87R$ the ASM C_t values agree with the blade resolved simulations extremely well at 8° and 10°. At 12° there is a small overprediction of approximately 5%. Whilst the Flamenco results are also very similar to the other solvers outboard from $0.4R$, they display more fluctuation across the span. An outboard rise in C_t is captured by all the simulation methodologies, starting between $0.86R$ and $0.9R$, although this feature is less prominent in the Flamenco simulation. The fall of C_t from $0.94r/R$ outwards is agreed upon by all of the solvers. The two solvers with data very close to the blade tip, HMB3 and Flamenco, both show a pronounced spike in C_t values in this location. The inboard behaviour of C_t varies significantly between solvers, however some form of increase in C_t is seen by all solvers except HMB3.

Figure 51 compares the sectional torque coefficient data of Helios, HMB3 and the ASM simulations. Overall, there is reasonable agreement between the methods, especially between $0.4r/R$ and $0.9r/R$. Both inboard and outboard of this region

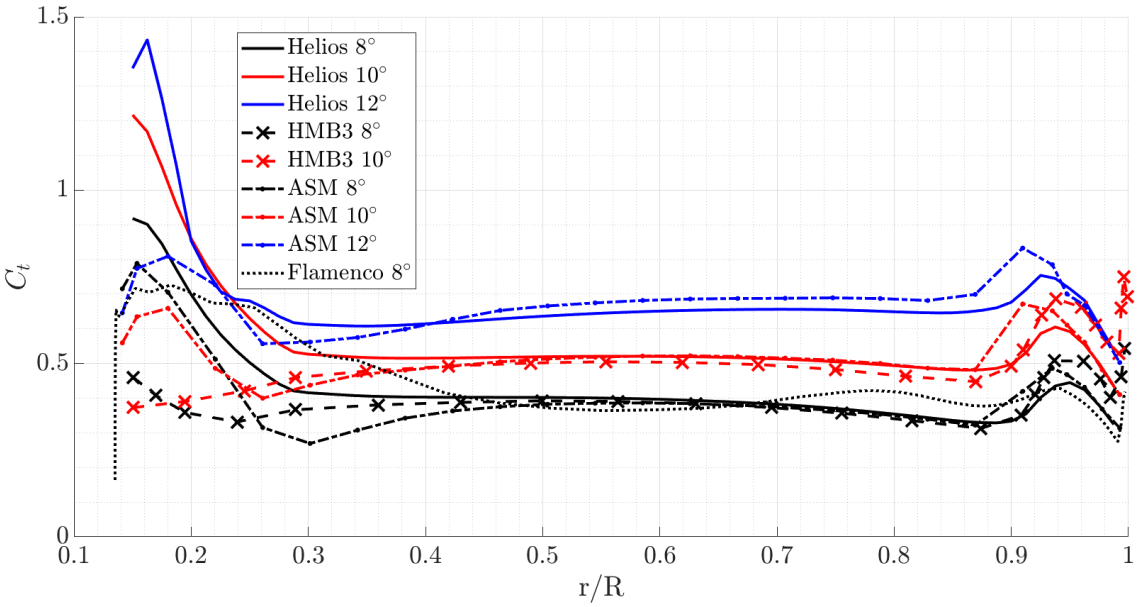


Figure 50: Comparison of sectional thrust values

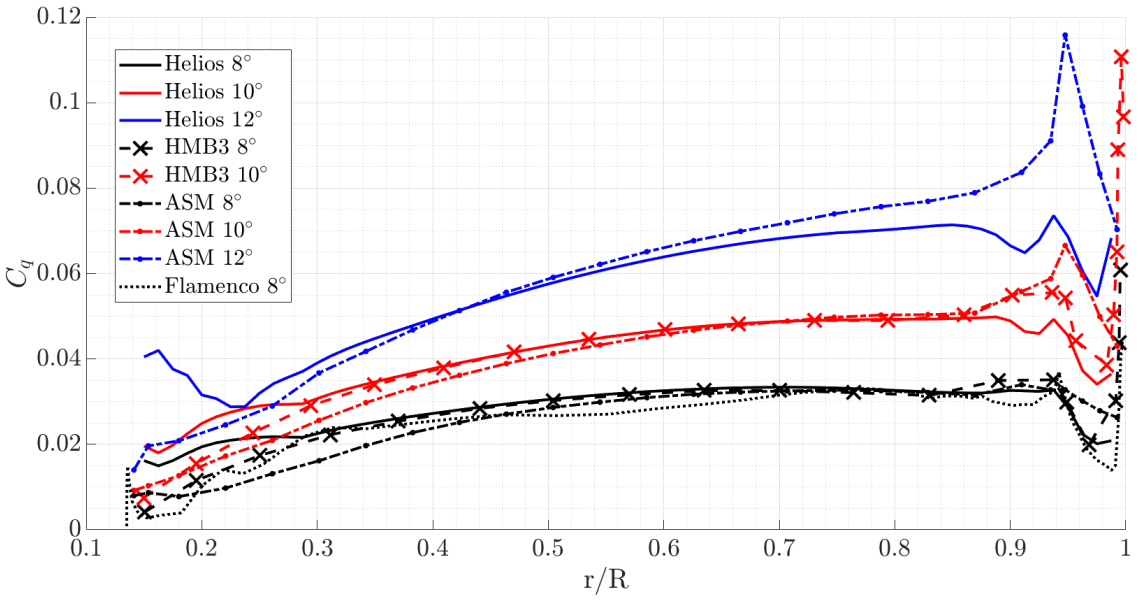


Figure 51: Comparison of sectional torque values

there is significant variation, although C_q reduces inboard almost universally. At 8° collective there is a decrease in C_q at approximately $0.95r/R$ across all solvers. Similarly to C_t, the Flamenco and HMB3 solvers display similar behaviour in the region very close to the blade tip. Overall, the high accuracy of the ASM in the middle of the blade, as well as the capturing of the general trends produced by the blade resolved methods, provide further evidence of the accuracy of the ASM. The Flamenco sectional loads also parallel the other solvers over much of the central portion of the blade, as well as matching the tip behaviour of the HMB3 solver.

6.6 Flowfield Visualisation

Figures 52, 53 and 54 compare the vertical flow velocity scaled by the blade tip speed for the Helios, Flamenco and ASM simulations. The Flamenco and ASM flowfields were sampled 4 rotations into the simulations. At this time, the integrated loads had reached an approximate steady state. However, the effect of the starting vortices are still visible in the central regions at the bottom of the images. Outside of this, the flowfields share a number of similarities. There is a contracting downwash below each rotor, with tip vortices being convected downwards along its edge. The tip vortex of age 90° is also clearly visible for all of the solvers just below the blade, shown by a peak downward velocity at $r/R = 0.9$, which is higher than the rest of the downwash of the rotor. It is also notable that the tip vortices older than 90° are generally larger and more diffused for the Flamenco and ASM simulations. This may be a result of the difference in when the flowfield was sampled. However, it could also be a reflection of the lower mesh densities for these simulations, as well as the lower order schemes used, particularly in the case of the ASM simulations. Helios and Flamenco also share very similar flow at the tip, with the spilling of the flow from the bottom to the top surface of the blade being prominent. As the ASM does not resolve the blade surface, the corresponding tip vortex is much weaker.

6.7 Summary of Hover Simulations

The simulation of the ARC rotor in hover displayed the accuracy of the methods used by the Flamenco flow solver. This includes the multiscale RBF mesh motion technique, which shows extremely small changes in orthogonality in the mesh close to the blade, with the minimum and global mean orthogonality changing by less than 0.5% within this region. Furthermore, the larger changes in grid quality were seen to generally occur further from the blade surface. The sufficiency of this grid deformation was confirmed with the simulation results being of comparable accuracy to other blade resolved simulations with regards to experimental data. The use of a wall function with a target first cell height of $y^+ \approx 100$ was also shown to give similar accuracy to the $y^+ \approx 1$ simulations. To the author's knowledge, simulations with such a large y^+ have not been performed for rotor load predictions. The ASM was also shown to have similar accuracy to the blade resolved methods in the middle of the collective range, but with poorer accuracy outside this range. With all solvers

displaying lower accuracy at the highest and lowest collective angles, some potential causes for these difficulties were presented. This includes large flow separation at high collective angles, and rotor-wake interactions at the low collective angles. Both of these phenomena are known to be difficult to accurately simulate and necessitate very computationally intensive simulations.

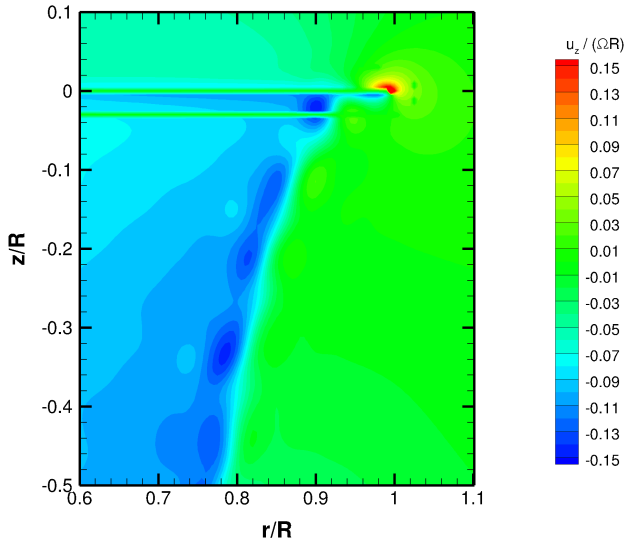


Figure 52: Scaled vertical velocity Helios

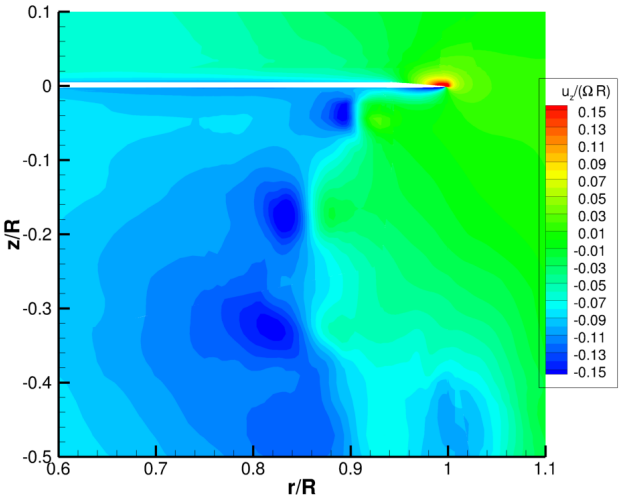


Figure 53: Scaled vertical velocity Flamenco

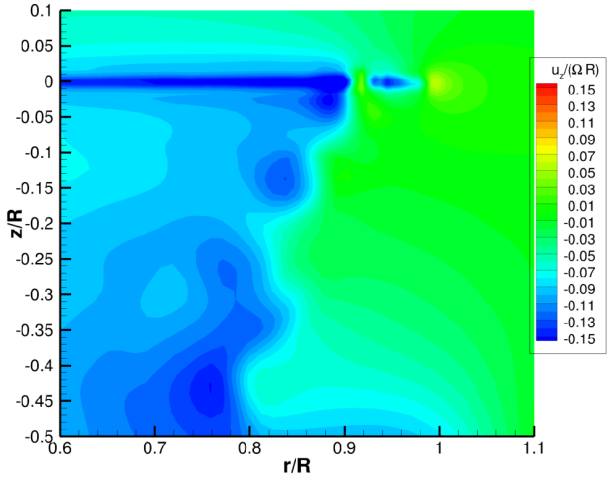


Figure 54: Scaled vertical velocity ASM

7 Forward Flight Simulations

This section presents the analysis of the forward flight simulations of the AH-64 rotor, with a particular focus on the comparison of the iterative greedy multiscale method. Firstly, the case description and meshes for the ASM and Flamenco are detailed. Secondly, the grid quality of the iterative, greedy multiscale method is compared to two other implementations of the multiscale method, one as described by Kedward et al. [2] and one with variable base node support radii. Different base node selection strategies are also explored. Following this, the preprocessing times are compared between the iterative, greedy, multiscale algorithms and a greedy, multiscale algorithm. The scaling of each sub-algorithm is compared, before comparing the total algorithms costs and reviewing the mathematical estimations provided in Section 4.5. Finally, the results of the aerodynamic simulations using the ASM and Flamenco are compared to flight test data and the blade-resolved codes Helios and HMB3.

7.1 Case Description

The atmospheric conditions used for the forward flight simulations correspond to test data published by Morris et al. [139], with a density altitude of 9460ft and average outside air temperature of 15°C. This corresponds to a pressure of 76 104.3 Pa and a density of $0.9201 kg/m^3$. The ratio of specific heats was assumed to be 1.4 and the Prandtl Number to be 0.7057. The rotor angular velocity was 290 RPM and the average C_T was 0.009031.

There are three flight conditions that can be validated against the literature corresponding to advance ratios $\mu_f = 0.2, 0.3$ and 0.35. The associated trim states were taken from Widjaja et al. [5] and are defined as per Table 19. α_s is the shaft angle, θ_0 is the collective pitch angle, θ_{1c} and θ_{1s} are the lateral and longitudinal cyclic pitch angles respectively, and β_0 is the coning angle.

Table 19: Trim states for the ARC helicopter in forward flight

μ_f	α_s	θ_0	θ_{1c}	θ_{1s}	β_0
0.2	-2.20°	9.26°	2.27°	-5.79°	3.13°
0.3	-4.19°	12.0°	1.78°	-10.0°	3.19°
0.35	-5.42°	14.7°	2.02°	-12.7°	3.20°

Where ψ_r is the blade azimuth angle, the total pitch angle is defined as:

$$\theta = \theta_0 + \theta_{1c}\cos(\psi_r) + \theta_{1s}\sin(\psi_r) \tag{171}$$

Simulations were initialised with the freestream velocities corresponding to the advance ratio and shaft angle.

7.2 Mesh Generation

7.2.1 ASM

The domains used for the ASM forward flight simulations were cuboid, with the boundaries perpendicular to the freestream flow being located $6R$ from the centre of the rotor. The front and rear boundaries were located $6R$ ahead and $12R$ behind the rotor centre respectively. The refinement regions of the forward flight mesh are similar to the hover mesh, but with the near-wake and far-wake regions being cuboid instead of cylindrical. The rotor region extended $0.3R$ above the rotor plane and $0.25R$ below it. The radius of this cylindrical region was $1.1R$ and the edge length in this region was $0.11c$. The parameters of the near-wake and far-wake regions are presented in Table 20. For this table, the x-axis is aligned with the freestream flow, the z-axis points upwards, and the y-axis forms a right handed system with these directions. The maximum values are the boundary locations in the positive axis direction, and minimum values are the boundary locations in the negative direction.

Table 20: Sizes of the near-wake and far-wake regions of the ASM forward flight simulation mesh

Mesh Region	$\mathbf{x_{max}}$	$\mathbf{x_{min}}$	$\mathbf{y_{max}}$	$\mathbf{y_{min}}$	$\mathbf{z_{max}}$	$\mathbf{z_{min}}$
Near-wake	5R	-1.1R	1.5R	-1.5R	0.41R	-1R
Far-wake	7R	-1.1R	1.75R	-1.75R	0.5R	-1.5R

The cell edge length was $0.21c$ in the near-wake region and $0.43c$ in the far-wake region. The cell length in the background mesh was $3.4c$. The forward flight meshes consisted of 11 977 114 cells. Figure 55 shows a slice though the mesh in the y-plane, displaying the geometry of the refinement zones.

The boundary conditions used in the forward flight meshes are shown in Table 21. Like the ASM hover simulation mesh, the hub is not represented in the mesh. The top and front boundaries are those in the positive x and z directions. The *freestream* boundary condition is a fixed value where there is an inflow, and zero gradient where there is an outflow. Similarly *freestreamPressure* is zero gradient over inflows and fixed value over outflows.

Table 21: Boundary conditions on the faces of the ASM forward flight meshes

Variable	**Top and front**	**Other**
$\mathbf{U}$	*freestream*	*freestream*
P	*freestreamPressure*	*freestreamPressure*
k	*fixedValue*	*zeroGradient*
ν_t	*fixedValue*	*zeroGradient*

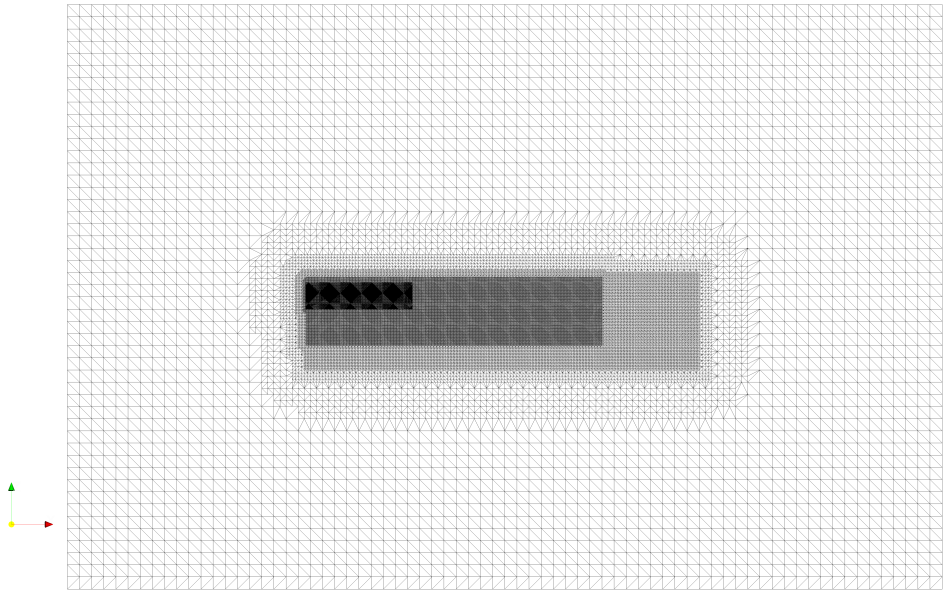


Figure 55: Slice through the ASM forward flight mesh

7.2.2 Blade Resolved Simulations

The mesh for the blade resolved simulations was developed from the quarter-blade mesh used for the hover simulations, with four quarters attached to each other to cover the full 360°. This means that all four blades and a full rotor hub are represented in the mesh. Due to computational restraints, it was not possible to run the simulations with four times the number of nodes, and therefore some modifications were required. The mesh density was generally reduced across the domain, focusing on reducing the far-wake density. The boundary-normal edge length of the cells on the blade surface was not changed. However, as the rotor wake is convected downstream and correspondingly does not extend far vertically, the upper and lower boundaries of the mesh could be brought closer to the blade. The forward flight mesh extended $4R$ above the rotor plane and $8R$ below the rotor plane. This reduced the impact of decreasing the number of mesh nodes. As farfield boundary conditions were used for the outer boundaries of the mesh, the wake structures should be allowed to dissipate before reaching the radial boundary. Correspondingly, the radius of the forward flight mesh was $9.83R$. Similarly to the hover simulations, the blades were modelled as no-slip walls, and the rotor hub as an inviscid wall. The outer boundaries were given fixed value boundary conditions with freestream conditions.

7.3 Comparison of Grid Quality Metrics

This section assesses the grid quality of the iterative, greedy multiscale method. The iterative greedy multiscale method is compared to two other multiscale RBF methods using the test case of the forward flight case as described in Section 7.1. Specifically, the deformation corresponded to the $\mu_f = 0.3$ trim case with blades at $\psi_r = 0°, 90°, 180°$ and 270°. This involves the correct pitch angle being applied to

each blade, as well as the constant coning angle. All methods used 720 base nodes for the deformation.

The first comparison method is the multiscale RBF method. As the base node selection for this method is unspecified, the base nodes were chosen as per the standard practice for research at The University of Sydney. This involves selecting the nodes at the corners of the surface mesh blocks. Amongst other uses, this method proved valid for the hover simulation collective angle adjustment. The RBF support radius used for the standard multiscale method was $1.5m$, which is approximately $2.8c$ and fits within the range of support radii presented in Table 11. As there is no motion at the hub, no base nodes were specified on its surface. However, as it has a defined position, the refinement nodes were used to stop it being moved by the blade motion. Due to memory limitations, the complete surface could not be specified as refinement nodes, so every 4^{th} point was used.

The second comparison method makes two scenario-specific changes to the standard multiscale method. The first of these is introducing the variable base node support radii as presented in Section 4.2. This meant that the hub was uninfluenced by the mesh motion, and could be removed from the base and refinement node system. This brings improvements in system conditioning, as proven in Section 4.2. Additionally, a different method was chosen for determining the base nodes. This method involved running the multiscale surface preprocessor on the surface mesh, and then selecting the first n points from the sorted list. As the surface preprocessor orders the points as to maximise the separation distances, this selection should give a highly uniform distribution of the base nodes across the object surfaces. This ensures that there is good coverage of the volume surrounding the blades. Furthermore, according to the investigation of system conditioning in Section 4.2, maximising the distance between the base nodes also reduces the system condition when compared to a closer clustering of the nodes.

The different base node selection methods are compared in Figure 56, showing the base nodes on the surface of the blade aligned with the x-axis. Figure 57 also shows a view of the nodes between $0.68R$ and $0.82R$, looking along the blade axis. Comparing the three distributions, the greedy method shows a distinct clustering of the base nodes at the leading and trailing edges of the blade. This corresponds to the parts of each blade section that experience maximum displacement from the pitching movement. Further, there is a small bias of the base nodes to be at the leading edge over the trailing edge. As the pitch angle is greater than zero in the greedy deformation state, the leading edge is correspondingly the part of each blade section which experiences maximal change in location.

Another trend shown by the greedy point selection is the clustering of points at the blade tip and, in particular, the blade root. The clustering at the blade root can be attributed to the reduced support radii for the nodes located closer to the root. Due to the smaller support radii, these base nodes generally exert less of an influence on the surrounding mesh volume. This is both due to more points being a distance

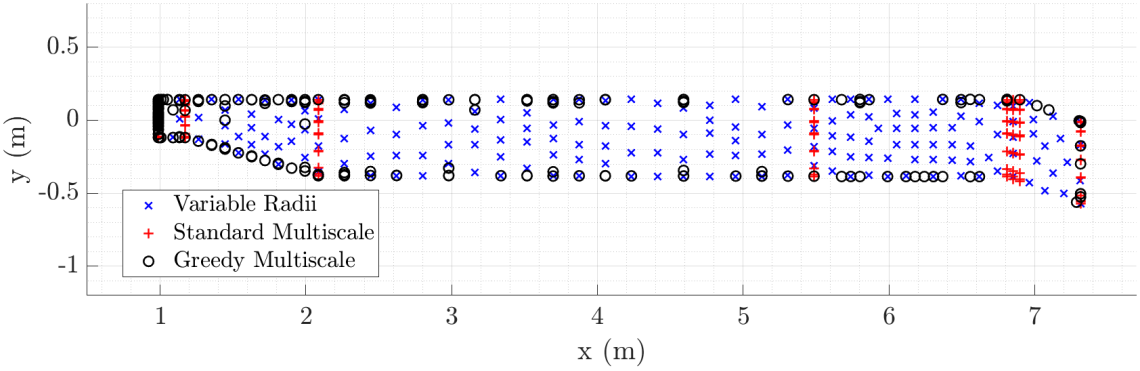


Figure 56: Comparison of base node location selections across methods

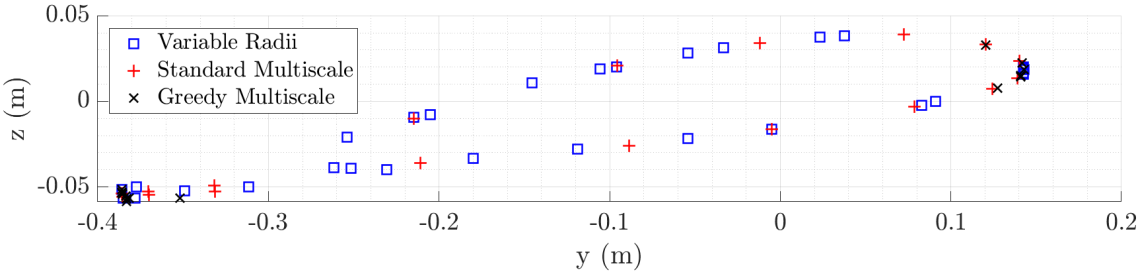


Figure 57: Comparison of base node location selections between $0.68R$ and $0.82R$ across methods

more than the support radius from the base node, as well as due to the scaled distance being larger for those points within the influence of the base node. The clustering at the blade tip is likely due to the movement of the mesh being greatest in this location, and hence requires stronger influences to maintain the mesh quality.

Table 22 shows the mesh quality in the near blade region using each of the three aforementioned methods. This shows the maximum i, j and k orthogonality components, the maximum and minimum node orthogonality q, and the mean orthogonality Q. As in Section 6.3, q_i, q_j and q_k are more orthogonal closer to 0, while q and Q are more orthogonal when closer to 1.

These orthogonality measures evidence the improved quality of the greedy multiscale method. It is noticeable that the multiscale and variable radii multiscale methods had maximum plane orthogonalities equal to 1, which would correspond to 0° or 180° angles being formed between mesh intervals along different curvilinear

Table 22: Mesh quality parameters within a distance of c from the blade for hover meshes

Method	$q_{i_{\max}}$	$q_{j_{\max}}$	$q_{k_{\max}}$	$q_{\min}$	$q_{\max}$	Q
Multiscale	1.000	0.9999	1.000	$2.972 \cdot 10^{-4}$	1.000	0.9395
Variable radii	1.000	0.9193	0.9363	0.2437	1.000	0.9598
Greedy multiscale	0.8751	0.8743	0.8625	0.3034	1.000	0.9608

coordinates. The minimum local orthogonalities are also distinctly improved by the greedy multiscale method. The minimum value of the standard multiscale method is almost 0, showing extremely poor mesh quality. The iterative greedy method also improves the $q_{\min}$ value by 24% over the variable radii multiscale implementation. The global average orthogonalities also support the improved mesh deformation quality of the greedy multiscale method, with the greedy multiscale Q being 2.3% higher than the standard multiscale method, and 0.11% higher than the variable radius method.

The orthogonality through sections of the forward flight meshes can be seen in Figure 58 to Figure 60. The most obvious difference between these plots is the decreased orthogonality across the slice of the mesh deformed using the standard multiscale method. This is particularly clear in the area ahead of the blade, where the orthogonality is approximately 0.1 less than the other two methods. It can be seen that this part of the mesh curves more sharply down towards the undeformed location, $z \approx 0$. This is due to the difference in support radius values, with the variable radius methods being able to employ larger support radii near the blade tip. The larger support radii result in lower curvatures between the deformed and undeformed mesh sections, as the influence of the base nodes decays over a greater distance. The differences between the variable radii multiscale method and the greedy multiscale method are not significant over this mesh slice. The mesh orthogonality is slightly higher in general for the greedy multiscale method, but by less than 1% over much of the slice. The difference ahead of the blade, where the displacement is greatest, is more noticeable.

Although the general mesh quality is important, it should also be noted that there are specific regions of untenable mesh quality in the multiscale and variable radii multiscale meshes. An example of this in the multiscale mesh is displayed in Figure 61, showing the outer faces of blocks adjacent to the blade surface at the root. Figure 61 shows a large shift of the volume mesh toward the blade tip, with a sharp curvature in the blade-normal direction to meet the blade surface at the specified RBF node locations. Furthermore, the mesh faces in the chordwise plane are highly curved, further reducing the mesh quality. Such a mesh would introduce an immense amount of numerical error to a simulation, not to mention that it may even result in an unstable calculation.

The variable radii multiscale method also suffers from some areas of very low quality deformation. An example of this is at the trailing edge root of the blade aligned with the x-axis. Figure 62 shows the mesh geometry in this region, with the outer blade surface marked in a thick outline. The most obvious problem with the volume mesh, marked with the purple lines, is the downwards curve of the edges extending in the negative x-direction from the rearmost edge of the blade. Additionally, the volume mesh edge highlighted in red should be approximately parallel to the edge highlighted in green, instead of running in the chordwise direction, as can be seen below the blade. The less rigid deformation of the near-surface mesh, evidenced by these two issues, would affect the accuracy of the numerics in this

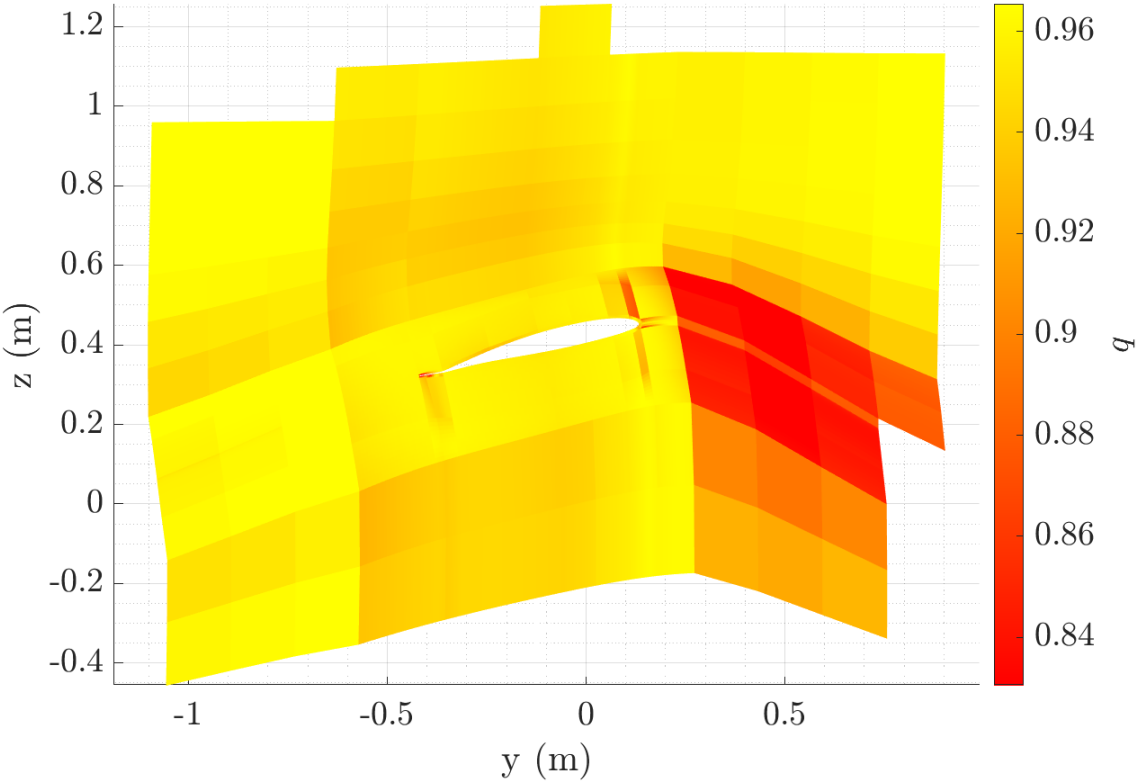


Figure 58: q in a slice at $x \approx 0.94R$, standard multiscale algorithm

portion of the mesh, degrading the solution quality.

7.4 Comparison of Preprocessing Times for Iterative and Non-Iterative Methods

This section analyses the scaling of computational costs for RBF simulation preprocessing, in which the iterative, greedy, multiscale method is compared to a noniterative, greedy multiscale method. The preprocessing tests are done for the forward flight simulations, specifically the $\mu_f = 0.3$ trim case. The displacement state for the greedy base node selection therefore involves the coning of the blades corresponding to the $\mu_f = 0.3$ trim state. The choice of pitching angle was not as simple, as it varies with the blade azimuth. The maximum pitching angle over the rotation was chosen, as this maximises the deformation quality at the state with maximum surface node displacement.

The analyses in this section suggested total costs were a function of the number of base nodes selected, and the size of the surface or volume meshes. Correspondingly, the scaling of the costs will be examined as a function of both base nodes selected, and of surface and volume node number as appropriate. The base node scaling was done with the full mesh as described in Section 7.2.2, which has 69 744 surface nodes and 84 540 volume nodes. The number of surface and volume nodes differ for two reasons. The first of these is that a surface node may correspond to multiple

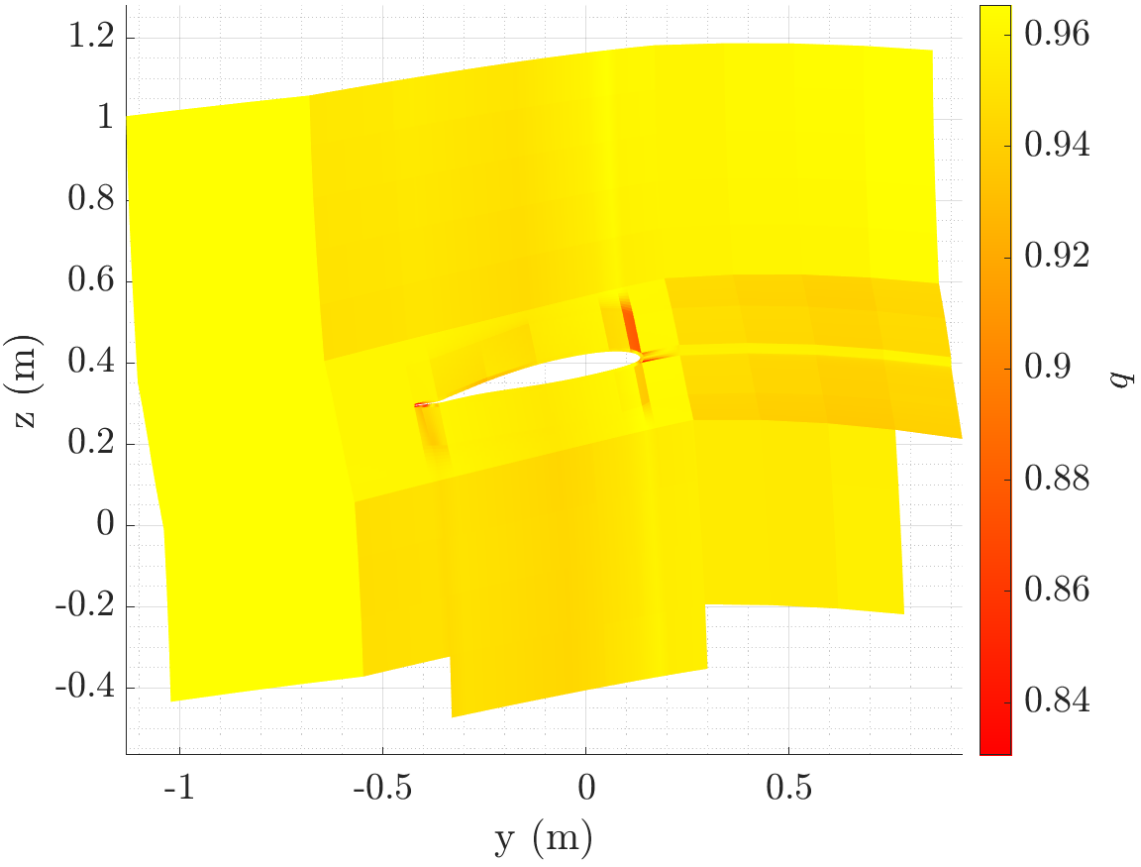


Figure 59: q in a slice at $x \approx 0.94R$, variable radii multiscale algorithm

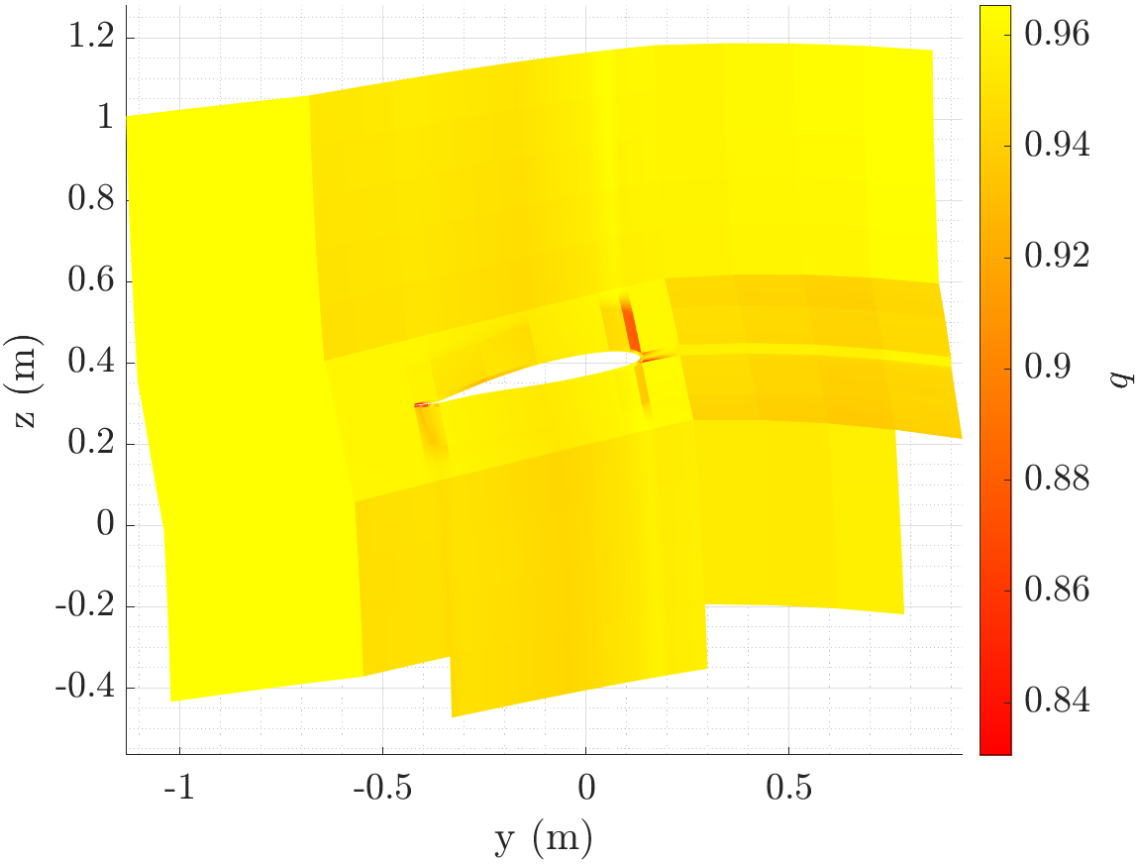


Figure 60: q in a slice at $x \approx 0.94R$, iterative greedy multiscale algorithm

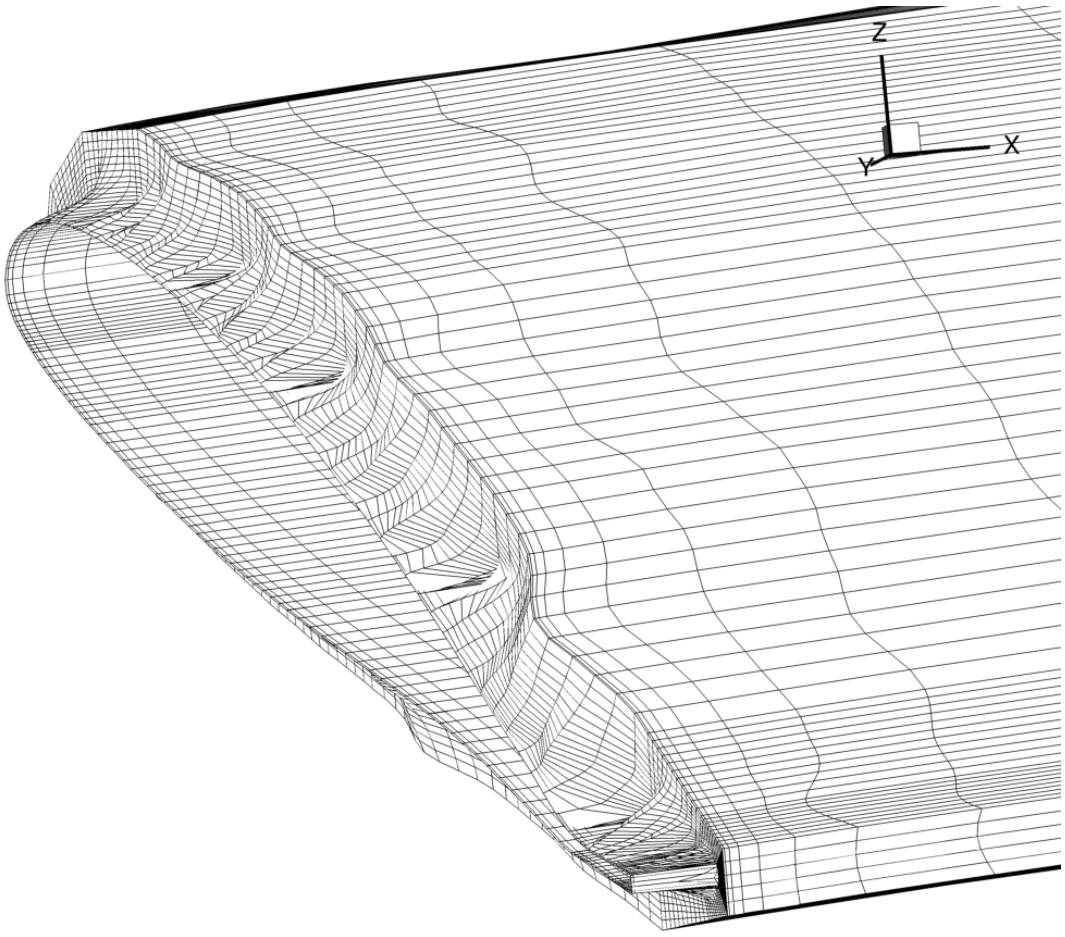


Figure 61: Deformed blade root mesh with standard multiscale RBF method, blade edges highlighted with thick lines

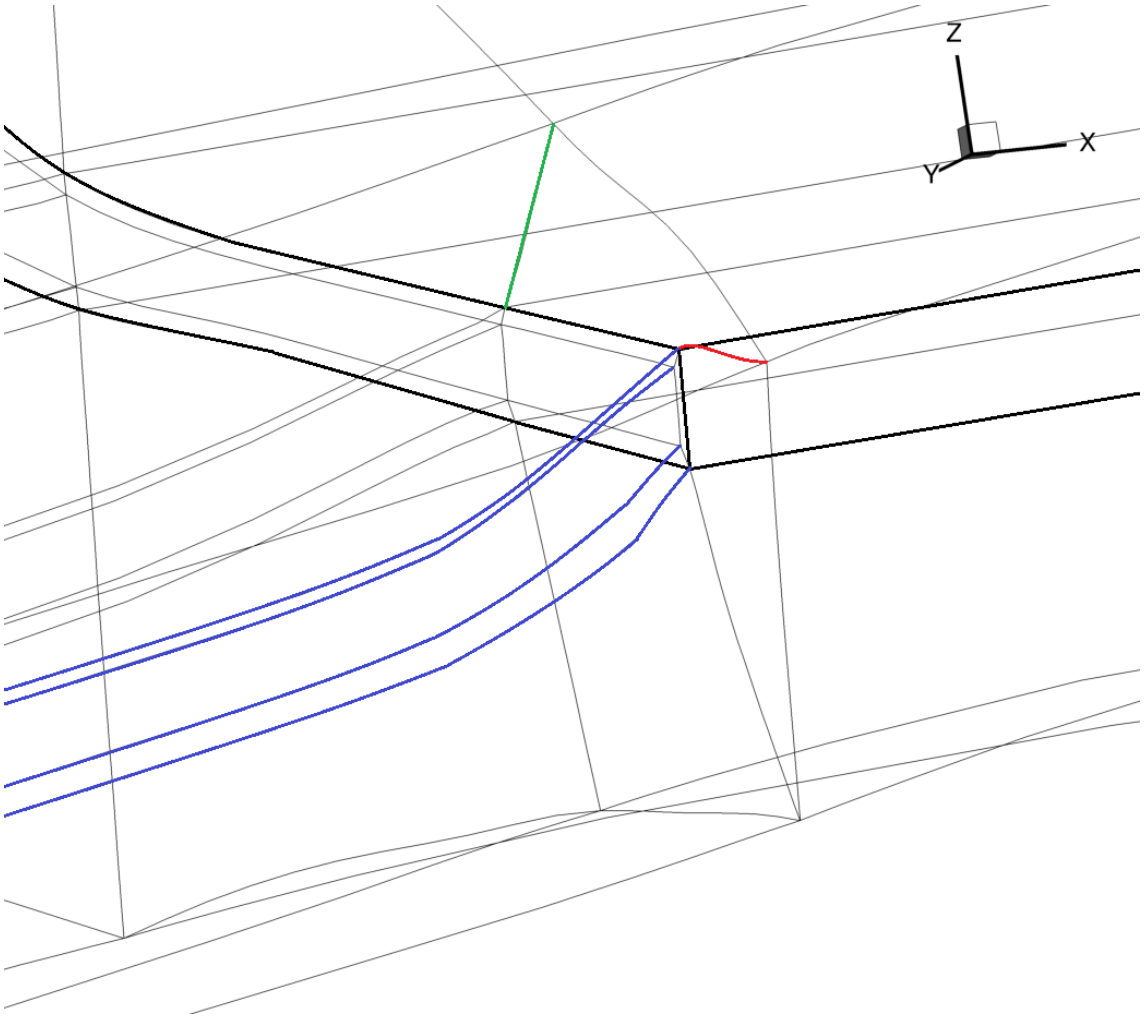


Figure 62: Deformed blade root mesh with standard multiscale RBF method

volume nodes. This could occur along edges of the blade, such as the trailing edge, blade root or blade tip, where multiple mesh blocks meet. Another reason for the discrepancy is that the complete blade surface is stored on each processor during a parallel execution of the code. In contrast, the mesh blocks are split between the processors, and therefore, the same volume node which is on a block boundary may be stored on two different processors. The timing of the preprocessing methods with regards to the volume and surface node number is done with a constant number of 1000 base nodes selected. The number of volume and surface nodes was varied by selecting a percentage of the nodes from the full mesh. This was done locally on each processor. Timings were collected at sizes of N_s and N_v ranging from approximately 10% of the full mesh numbers up to 100% at intervals of approximately 10%.

The comparison between methods is achieved through a comparison of cumulative runtimes, and through fitting an equation of the form ax^b to the cumulative algorithm costs against the relevant scaling variable. The curve fitting allows for an approximation of how each algorithm scales with the number of iterations and sizes of meshes. This can be used as an approximation of the asymptotic scaling of the algorithms. It should be noted that the operations which dominate the asymptotic order of the algorithms (as $n \to \infty$) may not dominate at realistic system sizes. Therefore, the exponent of the fitted curve is a better representation of the scaling of the algorithms in real-world applications.

The following graphs plot the cumulative runtime of the iterative method in blue and the cumulative runtime of the noniterative methods in dark green. The ax^b fits are plotted with dotted lines in cyan and light green respectively. The value of b for these fits are shown in the legend next to their respective lines. The scale of the time axis is different between each graph. The surface and volume node scaling graphs are similarly structured, but with the number of surface or volume nodes along the horizontal axis. For the surface and volume node scaling, the number of base nodes selected was kept constant at 1000. For the base node scaling, the preprocessors was run for a maximum time of 24 hours or until the memory limit on the computational hardware was reached, whichever limit occurred first.

In Section 4.4, it was shown that there are additional costs in computing some of the inputs for the iterative algorithms. These additional costs include determining which refinement nodes can be copied from the previous iteration, the new order of the refinement nodes, and the tracking of the distances from the refinement radii to the base nodes. The time taken determining these variables was also tracked and distributed to the algorithms that used these objects. When multiple algorithms used one object, the cost was distributed evenly to each of these algorithms. As an example, the copyable refinement node list was used by both the iterative algorithms for $\mathbf{L}$ and $\mathbf{\Psi}_{\boldsymbol{r}}$. Correspondingly, half the time spent determining this list was added to the iteration times for these two algorithms.

7.4.1 Surface Preprocessor

The costs of the surface preprocessor algorithms over the number of base nodes are shown in Figure 63. It is notable that the cost of the iterative method is slightly higher than the noniterative method over the observed range. This varies between a maximum of $193.3s$ at $N_b = 562$ and minimum of $0.2084s$ at $N_b = 1042$. However, the data fit suggests that the iterative method is scaling more favourably than the noniterative method by a very small margin, with a difference in exponent of 0.05.

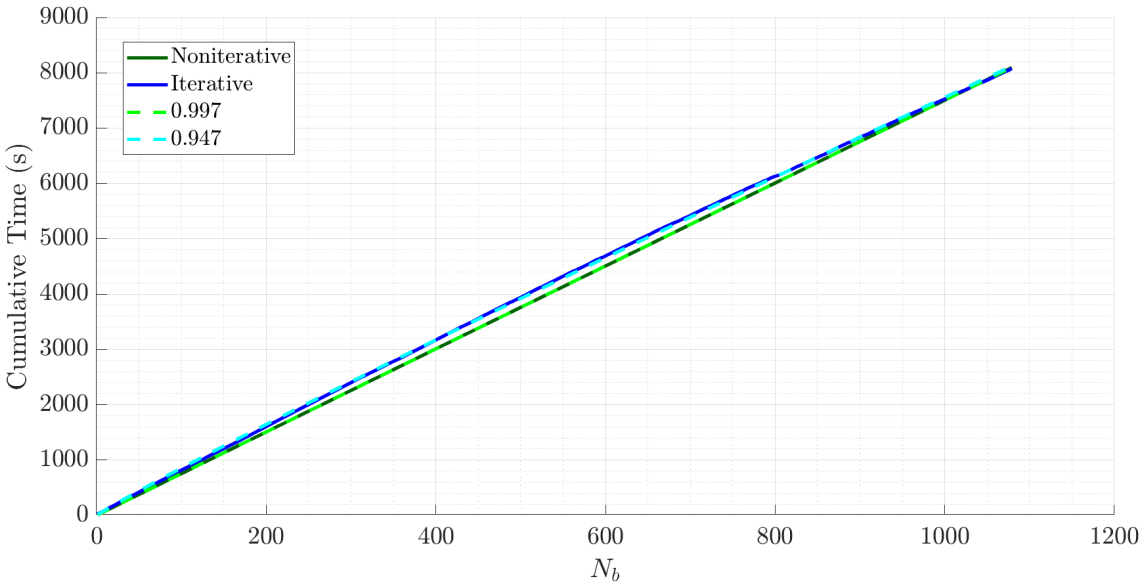


Figure 63: Cumulative time taken by the preprocessor in choosing N_b base nodes

Figure 64 shows the scaling of the preprocessor with respect to the number of surface nodes. This figure shows that the iterative methods were quicker over much of the range of surface sizes. The trend in surface scaling is likely a result of the differing N_p/N_r ratios with different sized systems. As explored in Section 4.5.1, the ratio of N_p/N_r tends to increase with the ratio N_{b_i}/N_s, and increasing N_p decreased the required calculations in that iteration. Therefore, when reducing the number of surface nodes, the 1000 base nodes become a higher fraction of N_s, and should therefore lead to quicker execution. While it may appear that the advantage of the iterative method decreases as N_s approaches 10 000, the percentage difference is actually larger in this region.

Table 23 compares the coefficients of the base node and surface node scaling fits to the big O asymptotic order. As predicted in the mathematical analysis, the iterative methods didn't significantly change the asymptotic scaling of the preprocessor, with the fit exponent only differing by just over 5% for N_b and just under 5% for N_s.

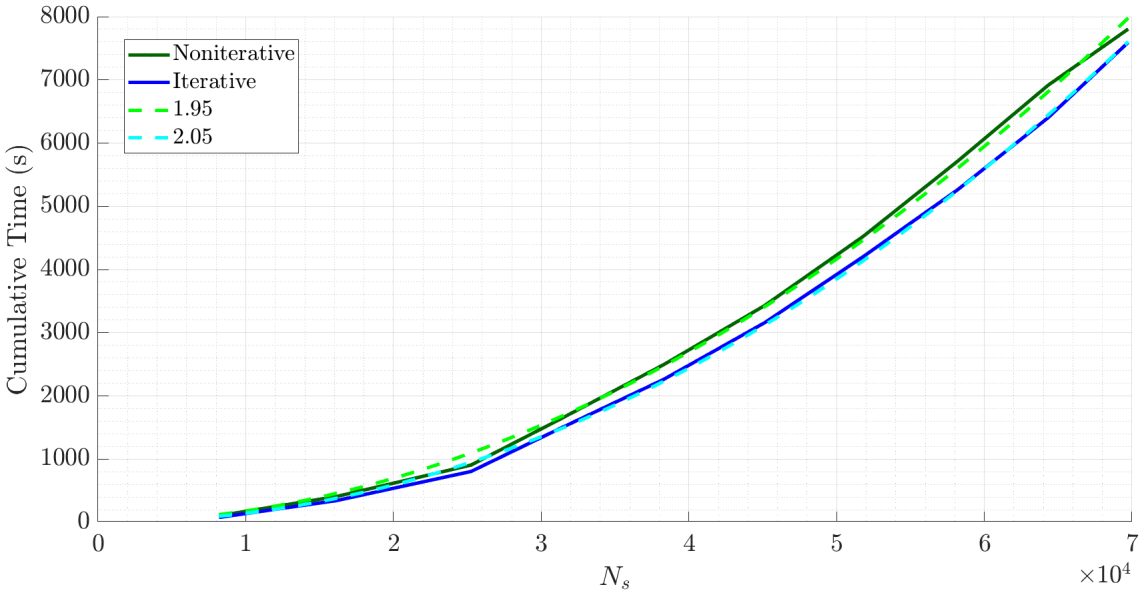


Figure 64: Cumulative time taken by the preprocessor in choosing 1000 base nodes over varying surface node numbers

Table 23: Comparison of preprocessor algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	7.64	0.997	1
N_b iterative	10.9	0.947	-
N_s noniterative	$2.783 \cdot 10^{-6}$	1.95	2
N_s iterative	$9.384 \cdot 10^{-7}$	2.05	-

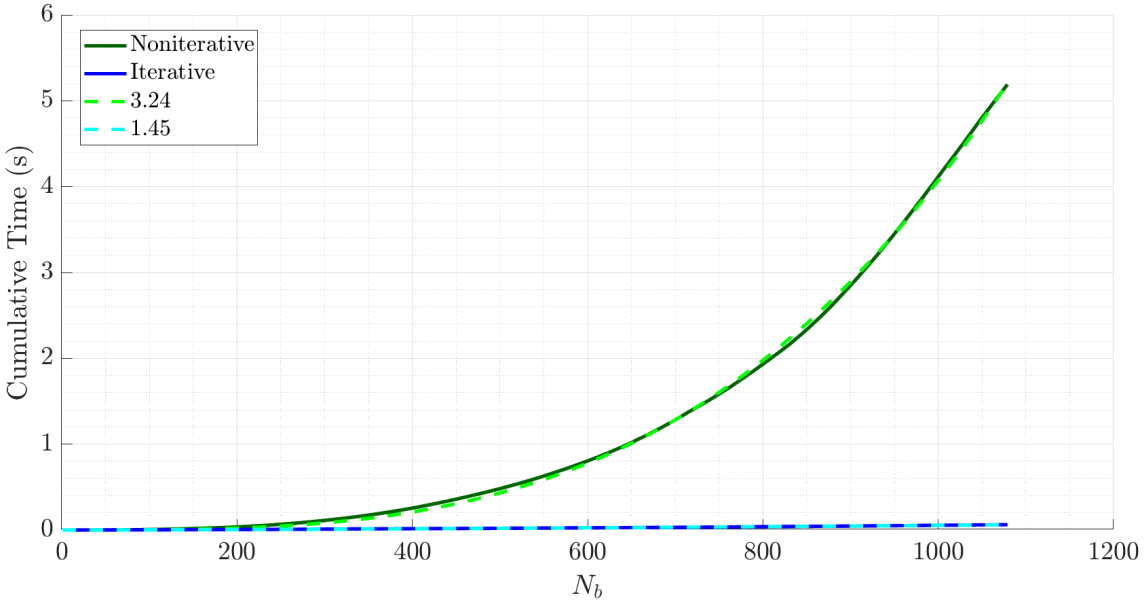


Figure 65: Cumulative time spent calculating $\mathbf{\Phi_b}$ in choosing N_b base nodes

Table 24: Comparison of $\mathbf{\Phi_b}$ algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	$7.905 \cdot 10^{-10}$	3.24	3
N_b iterative	$2.421 \cdot 10^{-6}$	1.45	2
N_s noniterative	1.95	-0.0732	0
N_s iterative	0.0794	-0.0321	0

7.4.2 Base Node to Base Node Influence Matrix $\mathbf{\Phi}_b$

Figure 65 shows the costs of the algorithms calculating $\mathbf{\Phi_b}$. The noniterative algorithm scales with a power of 3.237 and the iterative algorithm scales with a power of 1.450. The iterative methods show a very clear time saving in comparison to the noniterative method with much more favourable scaling.

The scaling of the iterative and noniterative algorithms with respect to the surface size is depicted in Figure 66. Whilst there is some variance over the range of surface sizes, the data fits show that the time spent on calculating $\mathbf{\Phi_b}$ is not a function of the surface size, as was expected from the mathematics.

Table 24 numerically compares the $\mathbf{\Phi_b}$ computation algorithms. Whilst the N_s scaling closely aligns with the asymptotic mathematics, the N_b scaling showed more deviation. Interestingly, while the noniterative method scaled less favourably than the big O cost, the iterative methods scaled more slowly.

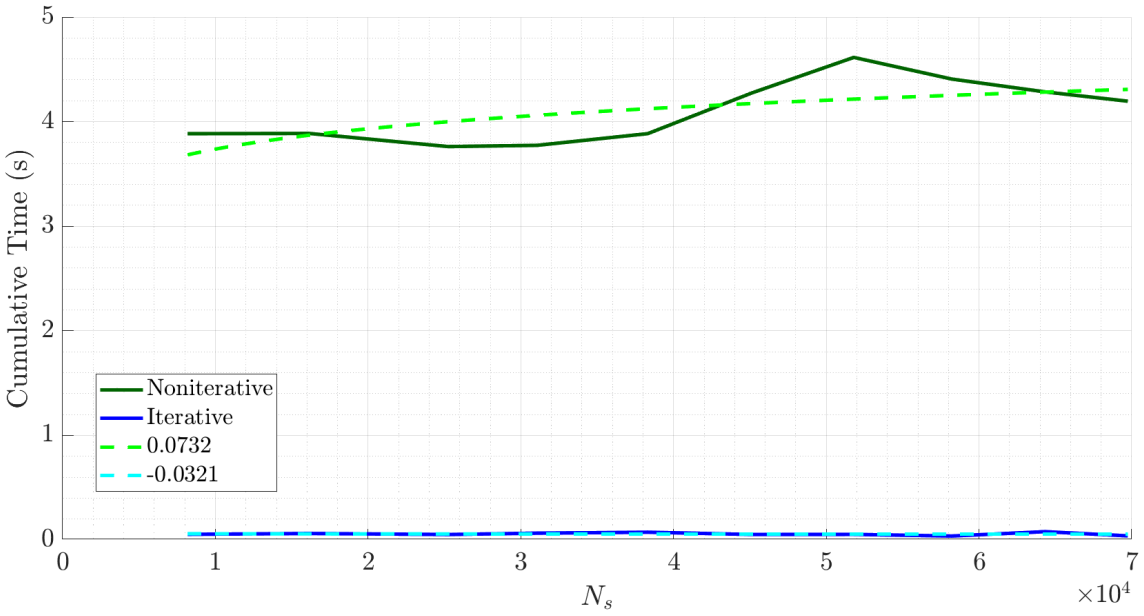


Figure 66: Cumulative time spent calculating $\mathbf{\Phi_b}$ in choosing 1000 base nodes over varying surface node numbers

7.4.3 Inverse Base Node to Base Node Matrix $\mathbf{\Phi}_b^{-1}$

The scaling of the $\mathbf{\Phi_b}^{-1}$ with respect to the number of base nodes selected is presented in Figure 67. Similarly to the $\mathbf{\Phi_b}$ algorithms, there is a very prominent difference between the noniterative and iterative algorithms. Compared to the other algorithms, it can be seen that the N_b scaling fit does not approximate the data as closely. It appears that the scaling of the noniterative inversion method changes at a few values of N_b. For example, the growth rate seems to increase after choosing approximately 900 base nodes. This may be the result of hardware related phenomena such as memory access speed, of the varying sub-functions present in the LAPACK code, or a combination of such factors.

Figure 68 demonstrates that the inverse base node to base node matrix algorithms became somewhat quicker as the N_s increased. This was not expected as the calculations are only a function of the base nodes. In light of this, there may have been unpredictable effects on the computation that were not accounted for in calculating the number of operations performed. Similarly to the base node scaling of the noniterative algorithm, these effects may include aspects such as memory access or other functions of the hardware.

Table 25 displays the numerical information regarding the data fits and big O scaling. The exponents seen in the data are quite distinct from the asymptotic values presented in Section 4.5, with the noniterative exponent being 34.3% more than the big O value when scaling with N_b, and the iterative value being 11.7% higher. The exponents of the N_s fits are both noticeably different to the asymptotic values as well.

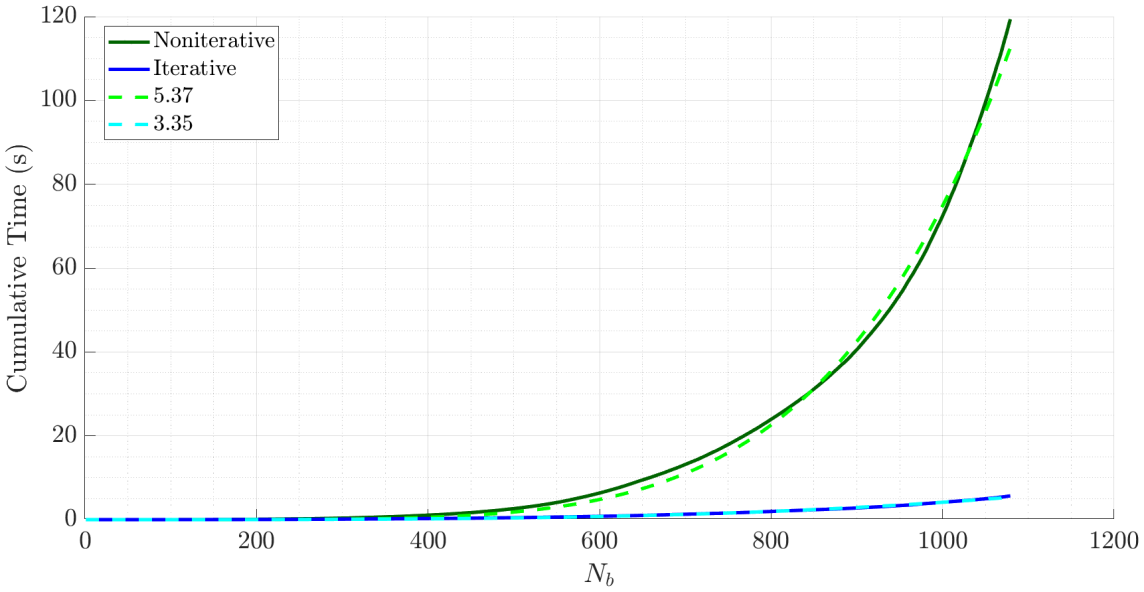


Figure 67: Cumulative time spent calculating $\mathbf{\Phi_b}^{-1}$ in choosing N_b base nodes

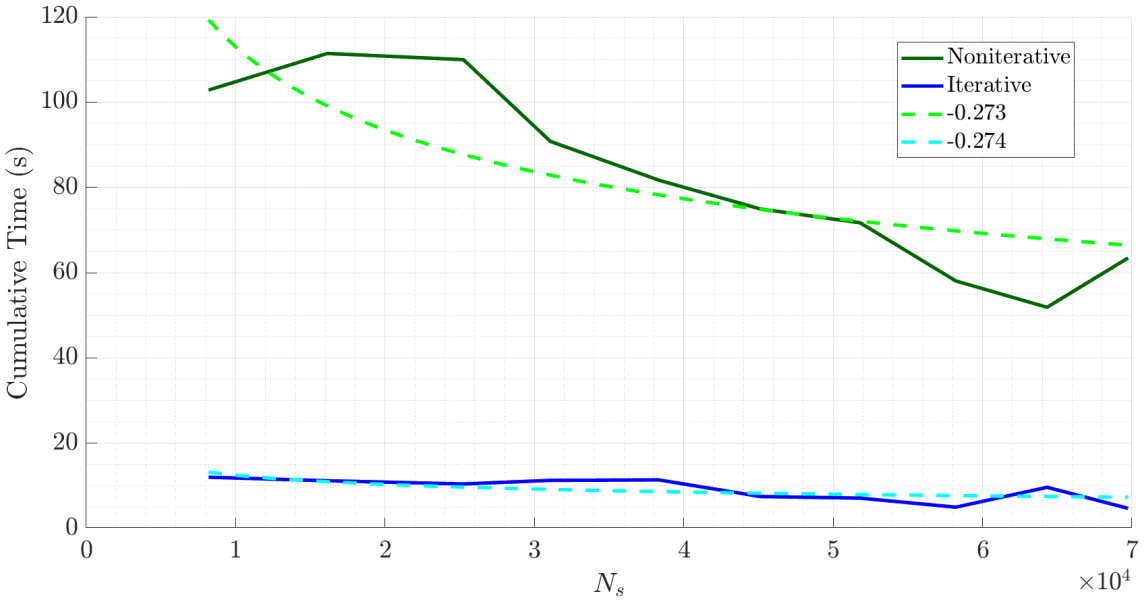


Figure 68: Cumulative time spent calculating $\mathbf{\Phi_b}^{-1}$ in choosing 1000 base nodes over varying surface node numbers

Table 25: Comparison of $\mathbf{\Phi_b}^{-1}$ algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	$5.763 \cdot 10^{-15}$	5.37	4
N_b iterative	$3.722 \cdot 10^{-10}$	3.35	3
N_s noniterative	1398	-0.273	0
N_s iterative	156.4	-0.274	0

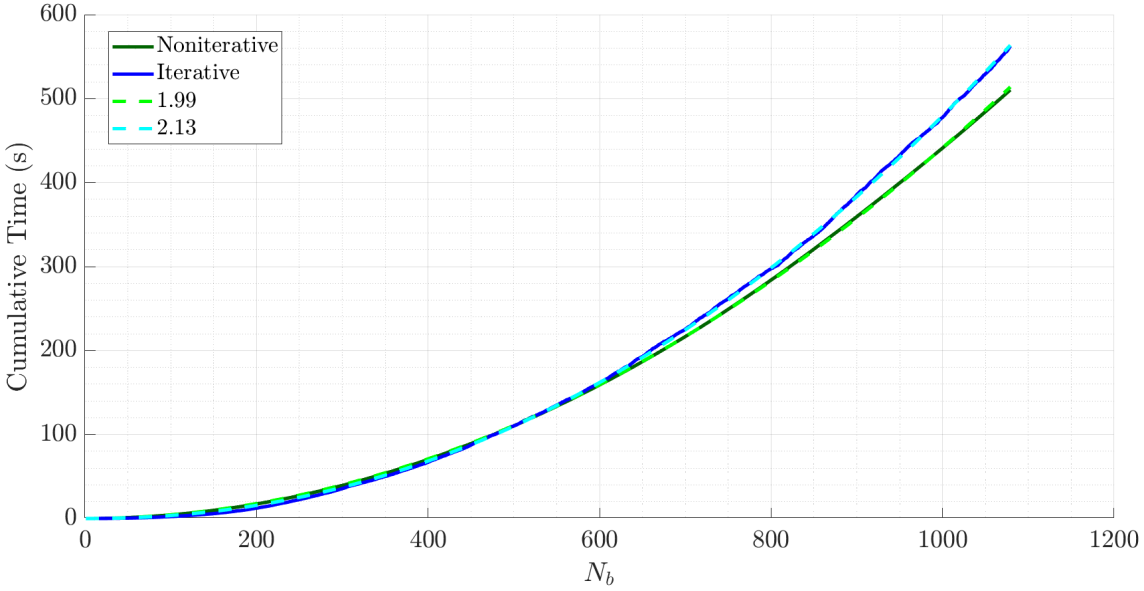


Figure 69: Cumulative time spent calculating $\mathbf{\Phi_r}$ in choosing N_b base nodes

7.4.4 Base Node to Refinement Node Matrix $\mathbf{\Phi}_r$

The time spent calculating $\mathbf{\Phi}_r$ over the range of base nodes can be seen in Figure 69. This graph shows a marginal advantage for the iterative method between $N_s = 3$ and 497, but being slower than the noniterative method beyond this point. While the number of RBF evaluations are definitely lower for the noniterative method, the reordering of $\mathbf{\Phi_{r_{i-1}}}$ necessitates a copy of this matrix, to and from which rows can be copied. The summation of these copies overpower the savings gained from reducing the RBF evaluations as the size of the matrix increases.

Figure 70 shows the scaling of the $\mathbf{\Phi_r}$ algorithms with surface node number. Like the base node scaling, the algorithms prove to be very similar. The iterative method tends to be slightly slower, but has a lower fit exponent. However, the difference in values is only approximately 4.5%, and may be skewed by the slower performance between approximately 3000 and 4000 surface points.

Table 26 shows the numeric data pertaining to the scaling of the $\mathbf{\Phi_r}$ algorithms. With the mathematical analysis not resulting in a viable big O asymptotic cost, the scaling of these algorithms in a realistic scenario can be seen. The approximately quadratic scaling with N_b and linear scaling with N_s is similar to the scaling behaviour of the $\mathbf{\Psi_b}$ algorithms, but is only valid when N_b is a small proportion of N_s.

7.4.5 Refinement Node to Refinement Node Matrix L

Figure 71 displays the scaling of the algorithms which calculate $\mathbf{L}$. There is a very large difference in costs between the iterative and noniterative methods, both in the magnitude of the costs and the scaling. Notably, the iterative algorithm takes less than 2% of the time that the noniterative algorithm does to select 1000 base nodes.

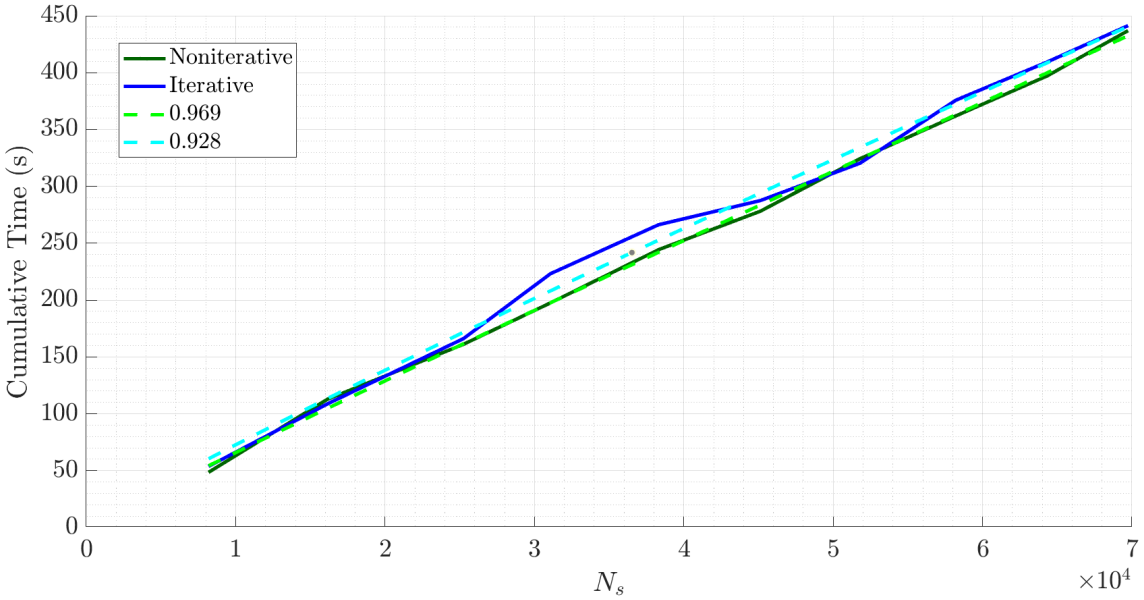


Figure 70: Cumulative time spent calculating $\mathbf{\Phi_r}$ in choosing 1000 base nodes over varying surface node numbers

Table 26: Comparison of $\mathbf{\Phi_r}$ algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	$4.59 \cdot 10^{-4}$	1.99	-
N_b iterative	$2.02 \cdot 10^{-4}$	2.13	-
N_s noniterative	$8.77 \cdot 10^{-3}$	0.969	-
N_s iterative	0.0142	0.928	-

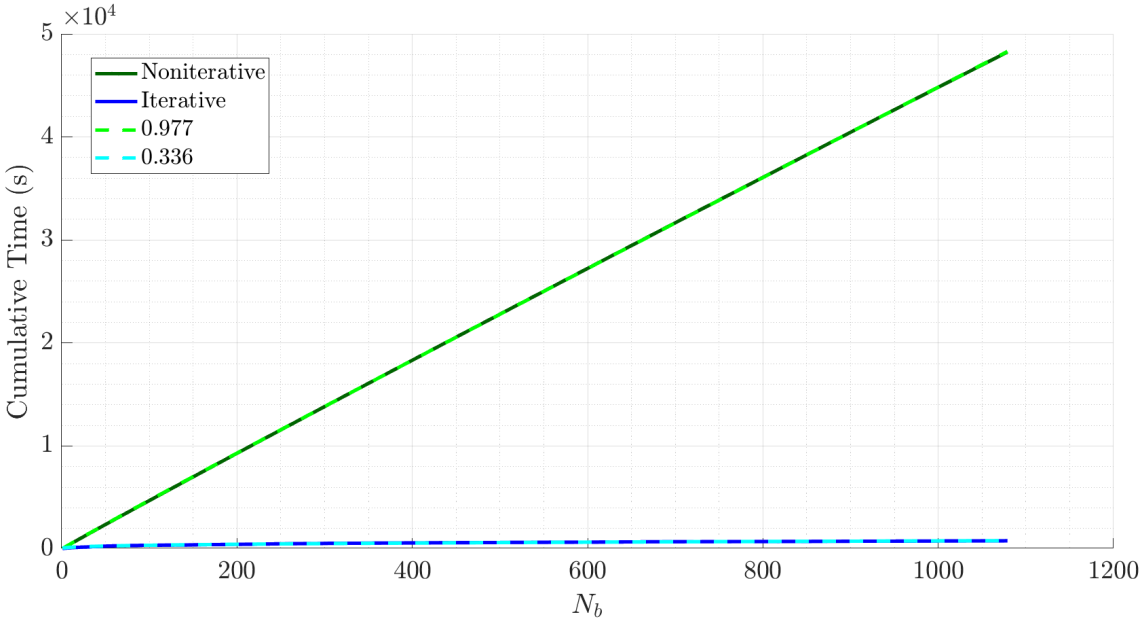


Figure 71: Cumulative time spent calculating $\mathbf{L}$ in choosing N_b base nodes

Table 27: Comparison of $\mathbf{L}$ algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	52.7	0.977	-
N_b iterative	77.4	0.336	-
N_s noniterative	$3.251 \cdot 10^{-6}$	2.09	2
N_s iterative	$1.509 \cdot 10^{-10}$	2.61	-

The increased speed of the iterative algorithm in calculating $\mathbf{L}$ is also seen in Figure 72, showing the surface node scaling. Although the exponent for the iterative fit is higher than the noniterative method by 0.52, both the absolute and percentage difference between the two methods grows in favour of the iterative method over the whole of the observed range.

Table 27 shows the data relating to the $\mathbf{L}$ algorithms. Whilst there were no formal big O costs for these algorithms in N_b, they both resulted in exponents lower than one. This was expected due to the growing size of N_r (the number of points on which the algorithms operate) with increasing N_b. The roughly quadratic scaling with N_s was also confirmed for the noniterative algorithm. Furthermore, the complete data confirms the analysis that, despite the worst case costs of the iterative algorithm being equal to the noniterative algorithm, the real costs are notably lower.

7.4.6 Base Node to Volume Node Influence Matrix $\mathbf{\Psi}_b$

The scaling of the $\mathbf{\Psi}_b$ algorithms as N_b increases is shown in Figure 73. Over the observed interval, the iterative methods are faster than the noniterative methods,

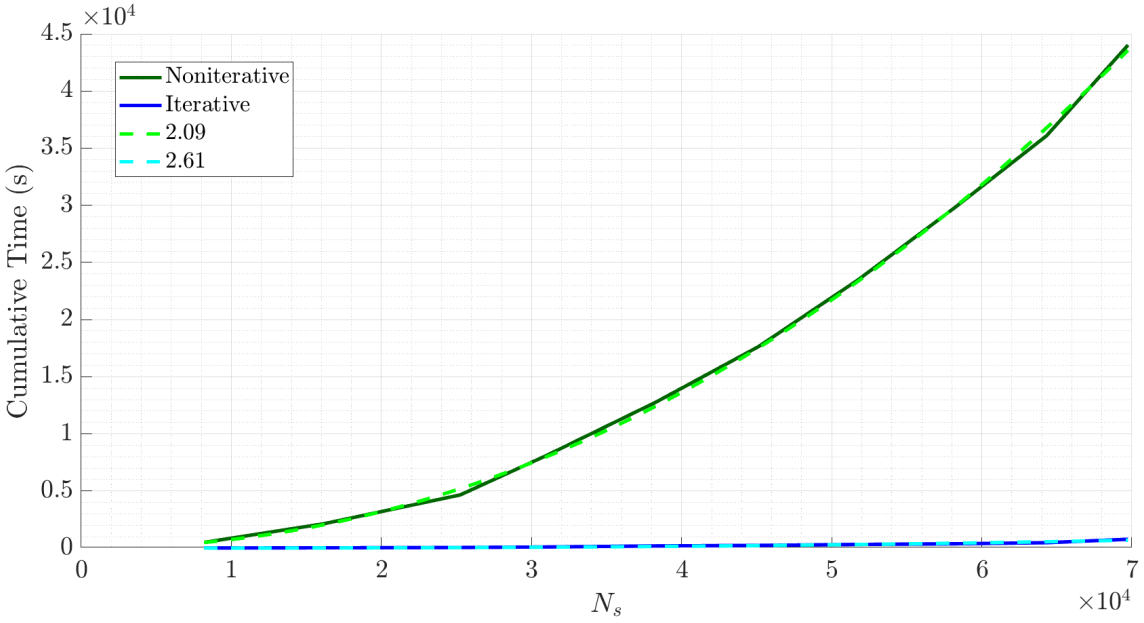


Figure 72: Cumulative time spent calculating $\mathbf{L}$ in choosing 1000 base nodes over varying surface node numbers

Table 28: Comparison of $\boldsymbol{\Psi_b}$ algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	$6.549 \cdot 10^{-6}$	1.97	2
N_b iterative	$1.701 \cdot 10^{-7}$	2.28	1
N_v noniterative	$1.36 \cdot 10^{-4}$	0.930	1
N_v iterative	$3.196 \cdot 10^{-5}$	0.919	1

with the difference between the two algorithms increasing.

The scaling of the $\boldsymbol{\Psi}_b$ with N_v is shown in Figure 74. While both algorithms grow sublinearly, the iterative algorithm shows a clear advantage with much better scaling.

The fit coefficient and big O orders for the $\boldsymbol{\Psi_b}$ algorithms are listed in Table 28. This data shows that the scaling of the iterative algorithm with N_b is worse than would have been expected from the asymptotic scaling. Additionally, both the a and b coefficients pf the N_v scaling fit are lower for the iterative methods, showing superior scaling in this regard.

7.4.7 Refinement Node to Volume Node Matrix $\boldsymbol{\Psi}_r$

The cumulative time taken calculating $\boldsymbol{\Psi_r}$ over the N_b range is shown in Figure 75. This figure shows a far superior outcome for the iterative algorithm, with the graph appearing similar to the scaling of $\mathbf{L}$ with N_b. This corresponds to the similarity of the algorithms for these two objects.

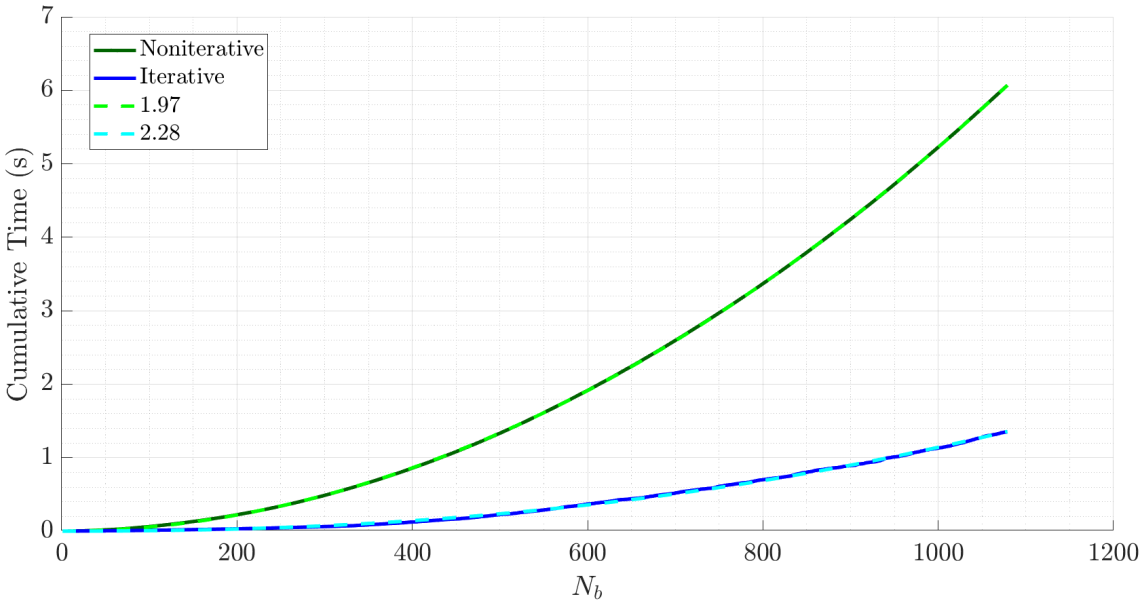


Figure 73: Cumulative time spent calculating $\boldsymbol{\Psi_b}$ in choosing N_b base nodes

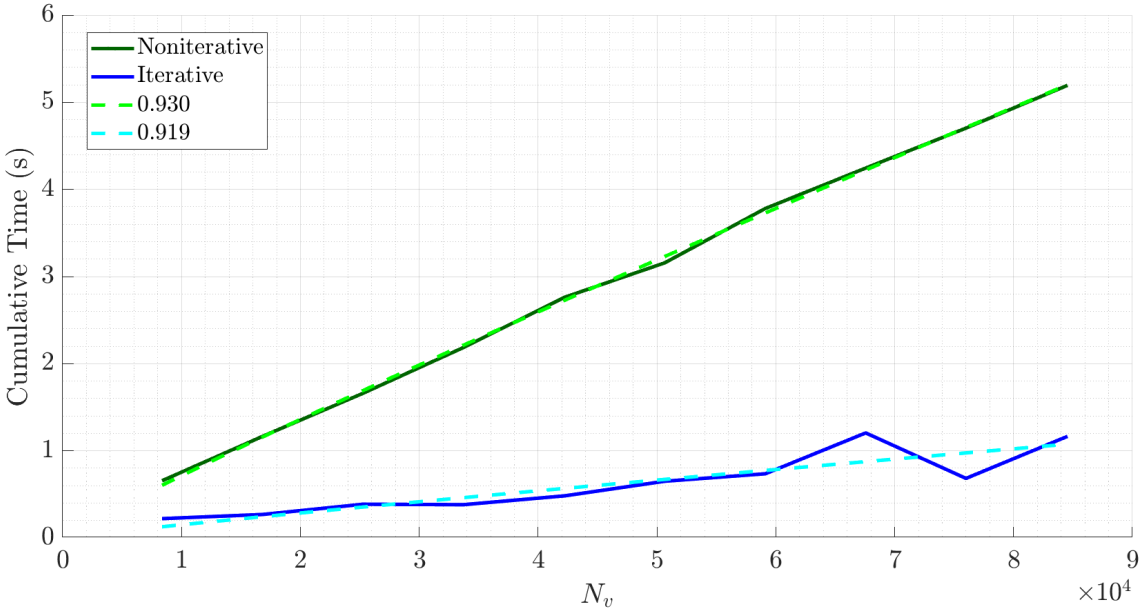


Figure 74: Cumulative time spent calculating $\boldsymbol{\Psi_b}$ in choosing 1000 base nodes over varying volume node numbers

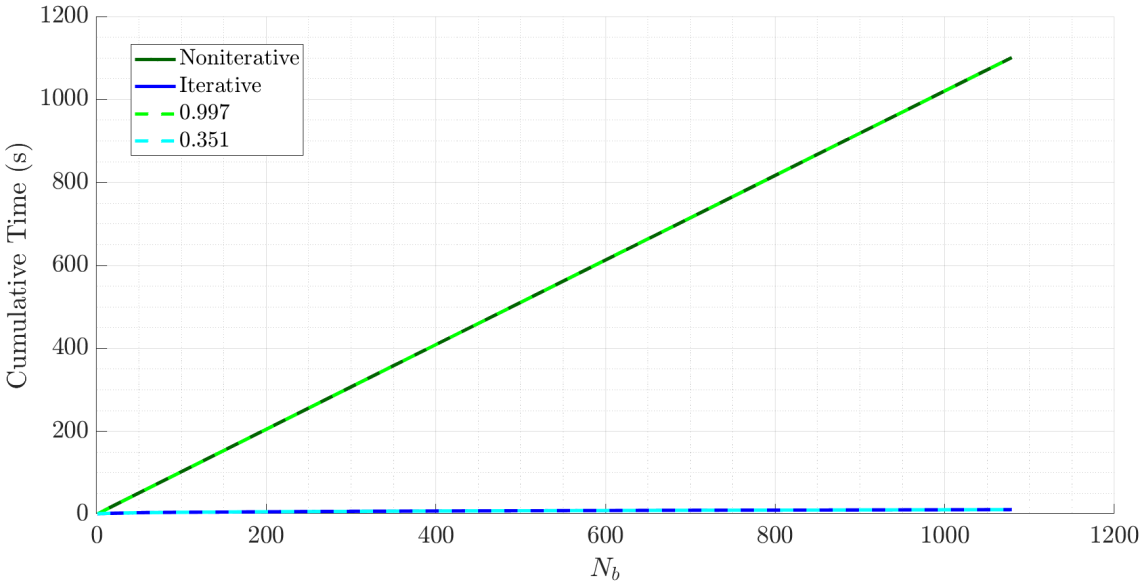


Figure 75: Cumulative time spent calculating $\boldsymbol{\Psi_r}$ in choosing N_b base nodes

The scaling of the $\boldsymbol{\Psi_r}$ algorithms with N_s and N_v is more complicated than the other objects, as this matrix calculates the interaction of the refinement and volume nodes. Considering that the number of refinement nodes is a function of the surface node number, the time taken by the $\boldsymbol{\Psi_r}$ algorithms is dependant on both N_s and N_v, as opposed to the other algorithms. Whilst the scaling of the algorithms could be calculated with respect to the surface and volume nodes independently, these would not be useful measures. The volume node and surface node numbers are closely correlated, as the volume nodes are a projection of the first layer of boundary layer nodes (see Section 4.6). It is possible to conceive of meshes with large disparities between the surface and boundary layer meshing, but these would require extremely unusual mesh structures and are not practical or realistic. Therefore, both the surface and volume node numbers were varied correspondingly when testing the scaling of the iterative and noniterative algorithms. Considering the surface mesh will typically first be defined, such that the necessary flow details can be resolved across the surface, and then the volume mesh is built off this surface, the number of surface nodes will be considered the base variable in this context. As such, Figure 76 shows the scaling of the $\boldsymbol{\Psi_r}$ algorithms with N_s. This graph shows that the iterative algorithm has far better performance over the range of surface nodes tested, with the difference between the methods growing.

Table 29 shows the coefficients for the scaling fits. This demonstrates that the iterative algorithm has both a lower a and b coefficient when scaling with N_b, and that the better performance over N_s is in spite of the higher exponent of the data fit.

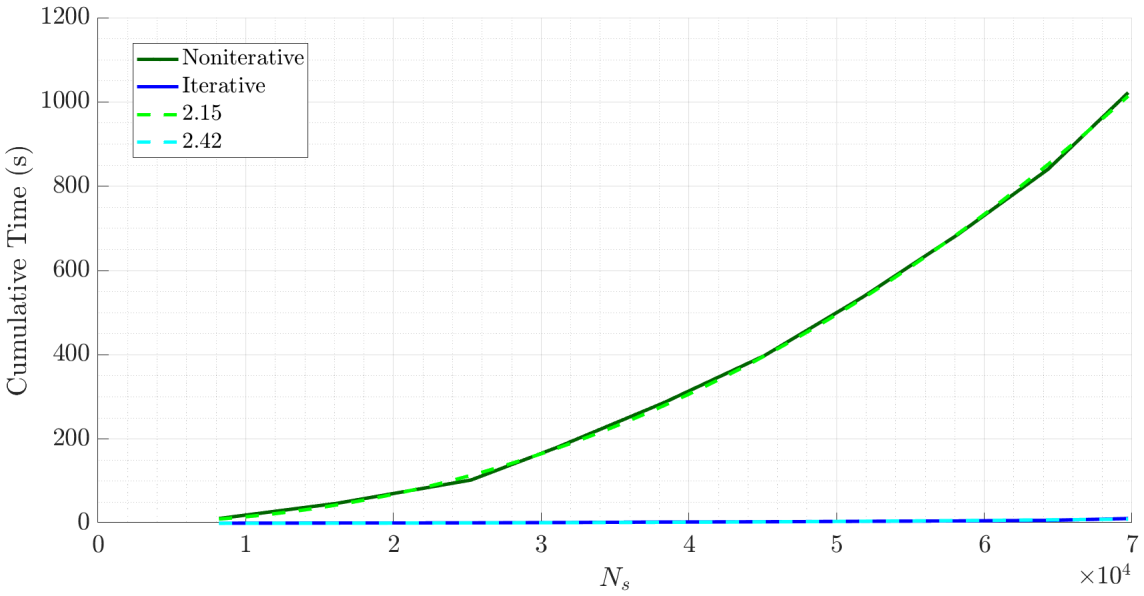


Figure 76: Cumulative time spent calculating $\boldsymbol{\Psi_r}$ in choosing 1000 base nodes over varying surface node numbers

Table 29: Comparison of $\boldsymbol{\Psi_r}$ algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	1.04	0.997	-
N_b iterative	0.975	0.351	-
N_s noniterative	$3.835 \cdot 10^{-8}$	2.15	-
N_s iterative	$1.92 \cdot 10^{-11}$	2.42	-

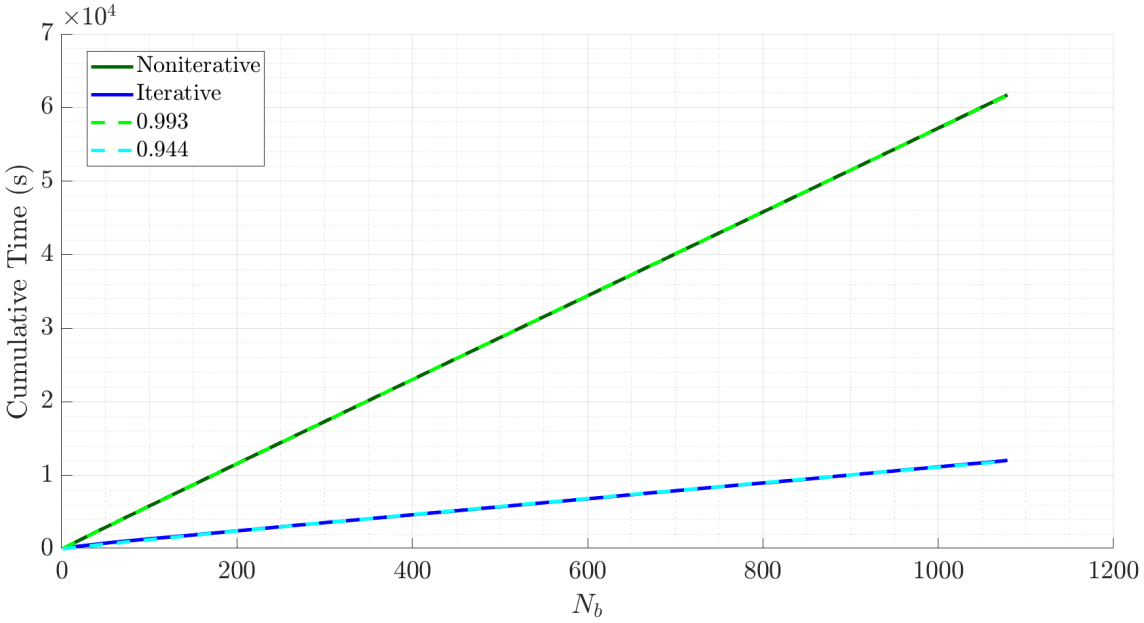


Figure 77: Cumulative time spent choosing N_b base nodes

7.4.8 Total Costs

Figure 77 shows the total cumulative times taken to choose N_b base nodes using the noniterative and iterative algorithms. It can be seen that the iterative method is clearly faster over the whole N_b range and scales more slowly with N_b. Both algorithms scale sublinearly, although the exponent for the noniterative method is less than 1% lower than 1. At $N_b = 1000$ the iterative method has taken 80.5% less time than the noniterative method. On the tests performed running on 256 cores, this corresponded to 17.1h for the noniterative method, and 3.12h for the iterative method.

Figure 78 displays the time taken by both algorithms to select 1000 base nodes over different surface sizes. Although the functions scale with both N_s and N_v, the surface node number will be used as the scaling variable, according to the reasoning discussed in Section 7.4.7. Both the iterative and noniterative methods grow close to quadratically with N_s, but the iterative methods grow more favourably than the noniterative methods.

Table 30 provides a numerical description of the total algorithm scaling. It can be seen that both fit coefficients are lower for the iterative method when considering the scaling with base node number. Examining the surface size scaling, the a coefficients differ by only 1%, but the exponent for the iterative methods is approximately 7.5% lower.

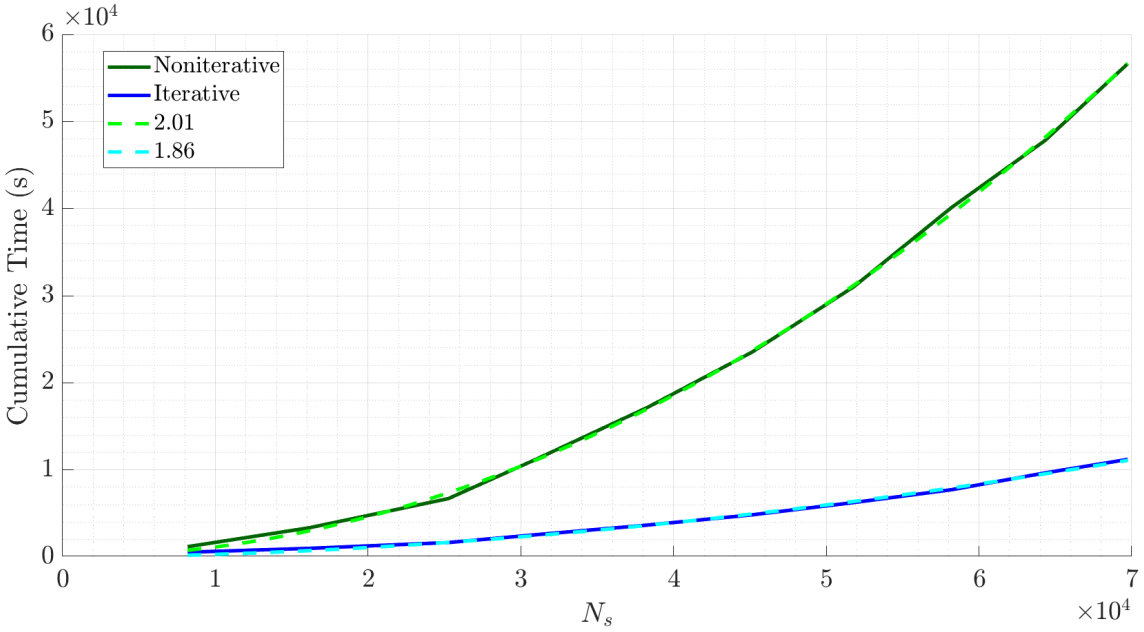


Figure 78: Cumulative time spent choosing 1000 base nodes over varying surface node numbers

Table 30: Comparison of total algorithm fit coefficients and asymptotic scaling

Algorithm	a	b	Asymptotic order
N_b noniterative	60.2	0.993	-
N_b iterative	16.4	0.944	-
N_s noniterative	$1.082 \cdot 10^{-5}$	2.01	-
N_s iterative	$1.094 \cdot 10^{-5}$	1.86	-

7.4.9 Overview of Algorithm Scaling

Tables 31 and 32 collate the data from the previous subsections for easier comparisons across all algorithms. It is clear that the iterative algorithm overall is successful in reducing the cost of choosing the base nodes. The fit exponents for the preprocessor, $\mathbf{\Phi_b}$, $\mathbf{\Phi_b}^{-1}$, $\mathbf{L}$ and $\mathbf{\Psi_r}$ are lower for the iterative methods than for the noniterative method. Whilst the exponents for the construction of $\mathbf{\Phi_r}$ and $\mathbf{\Psi_b}$ were greater for the iterative methods, analysing the graphs showed that the difference between the $\mathbf{\Phi_r}$ algorithms was minimal, and that the iterative $\mathbf{\Psi_b}$ algorithm was nevertheless significantly quicker than the noniterative method over the range of N_b observed. With the given fits for the $\mathbf{\Psi_b}$ algorithms, the iterative method would first be quicker than the noniterative method after choosing 130 144 base nodes. Considering this number is orders of magnitude more than the maximum N_b examined, the fits here may not be applicable, so the exact scaling is unknown. However, this is enough to speculate that the iterative method will be quicker for all realistic N_b sizes at the time of writing.

Despite the big O cost of the iterative algorithms with respect to N_b were $O(N_b^4)$ and $O(N_b^3)$ for the noniterative and iterative methods respectively, the scaling of these methods with realistic system sizes was shown to actually be sublinear. This is due to the relative dominance of the sub-algorithms. The big O costs were derived from the sub-algorithm with the highest order scaling, in this case generating $\mathbf{\Phi_b}^{-1}$. With realistic system sizes, it is seen that the time taken per iteration is strongly correlated with the number of points that are being operated on. This is shown in Figures 79 and 80, which display what fraction of each step of the multiscale greedy methods were spent doing which calculations. The data presented is a moving mean over 51 iterations. The calculations labelled "Other" references all computing time not spent in one of the sub-algorithms presented in this thesis. This includes actions such as solving the RBF system, updating the deformed volume node locations and determining the node with the maximum error criterion value.

The majority of the time in the noniterative method was spent preprocessing the surface nodes and calculating $\mathbf{L}$. These operate on intra-relations of the surface nodes and refinement nodes respectively. Additionally, N_r is generally very close to N_s as only a small proportion of the surface is chosen as base nodes. Thus, these algorithms will calculate the greatest number of relations, and correspondingly they occupy the majority of computing time. In contrast, despite the matrix inversion being more mathematically intensive, operating over the smaller number of base nodes means that these calculations are insignificant. A similar situation is seen in Figure 80, with the preprocessor taking up more than two thirds of the iteration time for most iterations. Computing $\mathbf{L}$ is also initially intensive, but quickly becomes a small part of the iteration time due to its very favourable scaling.

The scaling of the iterative methods is also favourable when considering the growth of the surface node number. The algorithms for $\mathbf{\Phi_b}^{-1}$, $\mathbf{\Phi_r}$ and Ψ_b had lower scaling coefficients, although the difference in exponents for the $\mathbf{\Phi_r}$ algorithms was only 4%. Many of the other algorithms shared a similarity to the scaling of the

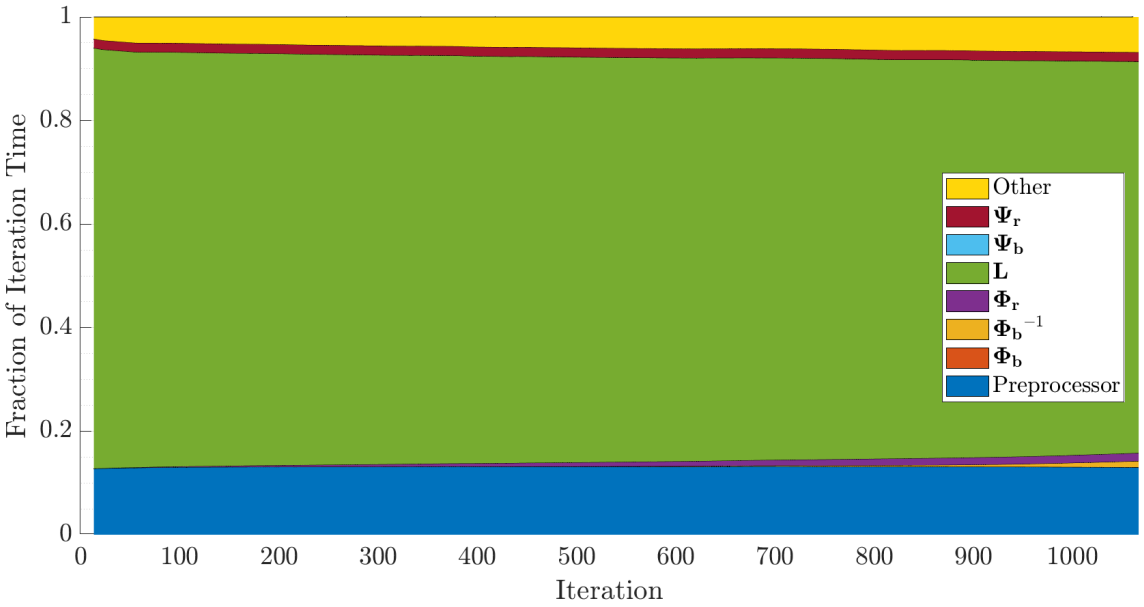


Figure 79: Moving mean of fraction of timestep spent calculating each object, non-iterative algorithm

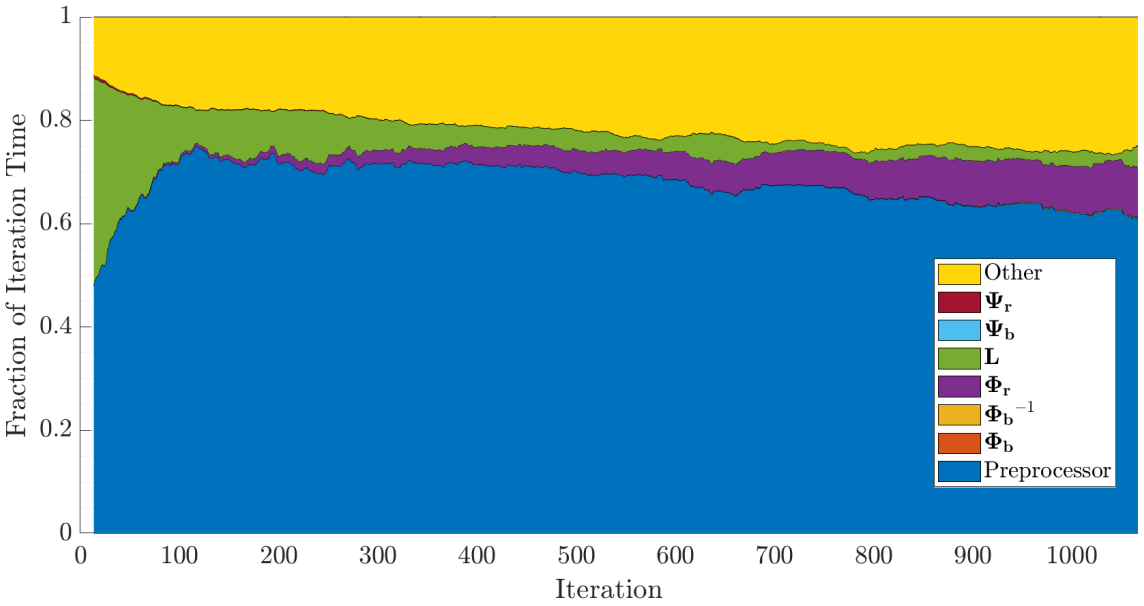


Figure 80: Moving mean of fraction of timestep spent calculating each object, iterative algorithm

Table 31: Comparison of fit coefficients and asymptotic scaling with N_b for iterative and noniterative algorithms

	a_{noniter}	a_{iter}	b_{noniter}	b_{iter}	$O()_{\text{noniter}}$	$O()_{\text{iter}}$
Preprocessor	7.64	10.9	0.997	0.947	1	-
$\boldsymbol{\Phi_b}$	$7.91 \cdot 10^{-10}$	$2.42 \cdot 10^{-6}$	3.24	1.45	3	2
$\boldsymbol{\Phi_b^{-1}}$	$5.76 \cdot 10^{-15}$	$3.72 \cdot 10^{-10}$	5.37	3.35	4	3
$\boldsymbol{\Phi_r}$	$4.59 \cdot 10^{-4}$	$2.02 \cdot 10^{-4}$	1.99	2.13	-	-
$\mathbf{L}$	52.7	77.4	0.977	0.336	-	-
$\boldsymbol{\Psi_b}$	$6.55 \cdot 10^{-6}$	$1.70 \cdot 10^{-7}$	1.97	2.28	2	1
$\boldsymbol{\Psi_r}$	1.04	0.975	0.997	0.351	-	-
Total	60.2	16.4	0.993	0.944	-	-

Table 32: Comparison of fit coefficients and asymptotic scaling with N_s or N_v for iterative and noniterative algorithms

	a_{noniter}	a_{iter}	b_{noniter}	b_{iter}	$O()_{\text{noniter}}$	$O()_{\text{iter}}$
Preprocessor	$2.78 \cdot 10^{-6}$	$9.38 \cdot 10^{-7}$	1.95	2.05	2	-
$\boldsymbol{\Phi_b}$	1.9	0.0794	-0.0732	-0.0321	0	0
$\boldsymbol{\Phi_b^{-1}}$	1398	156.4	-0.273	-0.274	0	0
$\boldsymbol{\Phi_r}$	$8.77 \cdot 10^{-3}$	0.0142	0.969	0.928	-	-
$\mathbf{L}$	$3.25 \cdot 10^{-6}$	$1.51 \cdot 10^{-10}$	2.09	2.61	2	-
$\boldsymbol{\Psi_b}$	$1.36 \cdot 10^{-4}$	$3.20 \cdot 10^{-5}$	0.93	0.919	1	1
$\boldsymbol{\Psi_r}$	$3.84 \cdot 10^{-8}$	$1.92 \cdot 10^{-11}$	2.15	2.42	-	-
Total	$1.08 \cdot 10^{-5}$	$1.09 \cdot 10^{-5}$	2.01	1.86	-	-

$\boldsymbol{\Psi_b}$ algorithms with N_b, with the cross-over point being at very high values of N_s (again, if the curve fits were still valid up to such large numbers). This includes the iterative method being faster for the calculation of $\mathbf{L}$ until approximately 215 million surface nodes, and until about 1.7 billion surface nodes for $\boldsymbol{\Psi_r}$. For a more practical extrapolation of the total cost, selecting 1000 base nodes on a surface mesh of 100 000 nodes would take 33.7h with the noniterative method and 6.04h using the iterative methods, which is a significant improvement.

The asymptotic orders of the sub-algorithms from big O analysis and the exponents of the data fits generally compare well, particularly for the noniterative algorithms. Only one noniterative algorithm differed from the asymptotic value by more than 10%, that being the scaling of $\boldsymbol{\Phi_b^{-1}}$ algorithm with N_b. The scaling of $\boldsymbol{\Phi_b^{-1}}$ with N_s also notably differed from the big O order by 0.273, but as this order was 0, there was no valid percentage difference. In contrast, only one iterative algorithm had a fit exponent within 10% of the asymptotic value. However, it should be kept in mind that there were fewer iterative sub-algorithms with valid big O orders. The difference in how closely the algorithms scaled to the asymptotic orders may reflect the relative complexities of the methods. The iterative methods with asymptotic costs were reliant on memory access to multiple objects from the previous iteration. These computational costs are ignored in an analysis of the number of operations

performed, but will impact the real runtimes of such algorithms.

7.5 Timestepping Performance

Given that the iterative greedy multiscale method is a development in the preprocessing of the multiscale method, there is no explicit difference in timestepping performance between these methods. If one allows different base node selection between the two methods, both in number and in location, there may be minor differences in timestepping performance due to the slight differences in volume node categorisation and evaluation. However, these will small differences over thousands of nodes.

While the timestepping performance is at best fractionally changed, a brief analysis of the additional overhead during timestepping is presented. The Flamenco forward flight simulation performed in this section was profiled over 26 304 timesteps, taking 11 hours. A total of 27.0% of the timestepping was spent performing mesh motion, with 24.3% being the RBF deformation and 2.57% rotating the mesh (the mesh rotation being achieved through multiplication of the nod coordinates by a rotation matrix). In comparison, 31% of the timestepping was spent calculating all boundary conditions (including mesh-internal block-to-block boundary exchange for halo cells), 16.9% was spent computing the inviscid fluxes, and 9.9% on viscous flux calculation. Within the RBF motion, the largest cost was the update of the halo cell nodes post deformation, with 9.67% of the per timestep solution computation. The next largest cost was calculating the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ matrices, being approximately 7.22% of the cost, of which the majority was spent on the calculation of $\boldsymbol{\beta}$.

7.6 Analysis of Simulation Results

Figure 81 graphs the integrated torque predictions of the blade resolved solvers and ASM against the advance ratio. Average values for ASM and Flamenco were averaged over half a rotation (two periods) after reaching a periodic steady state. As with the hover simulations, because there aren't flight test measurements at the exact advance ratios simulated, a curve was fitted to the experimental values to enable the evaluation of the error of the solver predictions. An approximation for the rotor torque in edgewise forward flight above approximately $\mu_f = 0.1$ is [9]:

$$C_Q \approx \frac{\kappa C_T^2}{2\mu_f} + \frac{\sigma C_{do}}{8}\left(1 + 4.65\mu_f^2\right) + \frac{1}{2}\mu_f^3\frac{f}{A} + \lambda_c C_T \tag{172}$$

As C_T is constant over the forward flight simulations, the form of the fitted curve is:

$$C_Q = \frac{a}{\mu_f} + b\mu_f^3 + c\mu_f^2 + d \tag{173}$$

With the coefficients a, b, c and d being tuned to fit the dataset. The percentage errors of the solvers in comparison to the flight test data fit are displayed in Table

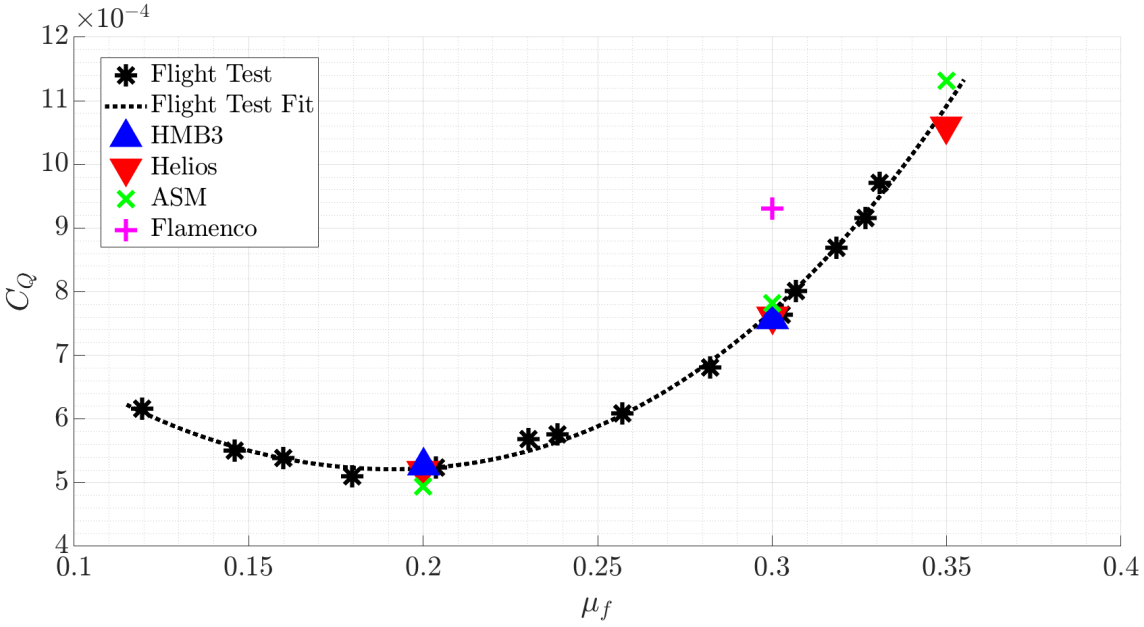


Figure 81: Comparison of integrated torque predictions over advance ratio range

33. The error between the ASM and experimental data is comparable to the blade-resolved solvers results at $\mu_f = 0.3$ and 0.35. In contrast, the error at $\mu_f = 0.2$ is multiple times the error of the blade resolved solvers. Unlike the hover simulations, the Flamenco accuracy of the forward flight simulations was poor with a 21% over-prediction.

The main candidate for the inaccuracy of the predictions relative to the hover simulations is thought to be the lack of mesh refinement. As the first cell thickness was not changed for the forward flight mesh, but the additional freestream velocity increases the maximum flow speed relative to the blade, the corresponding y^+ is increased. Using the same flat-plate estimation as for the hover mesh, at an advance ratio of 0.3, the same cell thickness now corresponds to an estimated y^+ of 125. As this is within the valid y^+ range of the Spalding wall model [140], this increase in y^+ is not an issue per se. However, it does mean that the number of cells resolving the boundary layer is reduced, which, despite the use of a wall function, should intuitively be less accurate. While this has not been a topic of deep study, there is some limited research that points to a reduction in accuracy with increased y^+ [141, 142]. In addition to the reduced boundary layer resolution, the reduced mesh density in the rotor plane will also resolve the turbulent and vortical wake structures less accurately and more diffusively. As shown in the hover simulations, wake structures can have a pronounced impact on the aerodynamics. Considering that the bulk of the rotor is operating in the wake of the other blades, a lack of resolution may have a notable effect on accuracy. One could also posit that the log-layer mismatch may also contribute to incorrect wake simulation, but the lack of mesh density would diffuse the solution to being largely indifferent to this phenomenon. Finally, there are notable areas of flow separation, especially on the retreating blade. As discussed in the hover simulation analysis, the empirical underpinnings of the Spalding wall

Table 33: Percentage error in C_Q between flow solvers and flight data fit

μ_f	ASM	Helios	HMB3	Flamenco
0.2	-5.51	-0.682	0.750	
0.3	1.62	-1.02	-1.70	20.9
0.35	3.63	-2.93		

model are not valid under pressure gradients above the viscous sublayer.

7.7 Summary of Forward Flight Simulations

This section explored the preprocessing times and grid quality of the iterative greedy multiscale algorithm. Firstly, the quality of the deformation using this method was proven superior to two alternatives: a standard multiscale method and a multiscale method with variable base node support radii. Since the base node selection is not specified for these methods, two strategies were chosen as part of the comparison. The standard multiscale method was coupled with selecting the nodes at the vertices of the blade surface mesh blocks, and the variable radii method selected nodes with the greatest separation distance, providing a more even coverage of the blade surface. When examining the orthogonality of the deformed meshes, the methods with variable base node support radii showed a clear advantage. The support radii were able to scale with the large deformations caused by the rotor coning, and hence the mesh curvature could be lower in the outer rotor area, increasing the mesh orthogonality. Areas with severe mesh curvature were also found in the standard multiscale and variable support radii methods, but weren't seen in the greedy multiscale mesh. These would have likely crashed the standard multiscale simulations, and would definitely reduced the accuracy of the flow simulation of the variable radii simulations.

When comparing the preprocessing times to a noniterative method, the scaling of most iterative sub-algorithms with regards to the number of surface nodes and to number of base nodes chosen was superior. The scaling of some sub-algorithms was approximately equal to the noniterative methods, but the total preprocessing cost was significantly shorter for the iterative algorithm. For example, choosing 1000 base nodes with approximately 70 000 surface nodes was 80.5% faster when using the iterative methods. This preprocessing only consumed just over three hours of wall time running on 256 cores, which is a small cost when compared to the total computational resources required for a high-fidelity simulation. This ultimately means that there is little cost to using this new method, which provides the advantages of nearly maximising the mesh deformation quality to base node number ratio, as well as automating the base node selection.

The aerodynamic simulation of the rotor in forward flight was undertaken stably with the new mesh motion algorithms. While the ASM was able to produce predictions of similar accuracy to the hover simulations, the Flamenco simulation overpredicted the integrated torque at $\mu_f = 0.3$ by 20%. The primary reason sug-

gested to explain this loss of accuracy is the inability to increase mesh resolution relative to the hover mesh, a result of computational restraints. This effectively increased the y^+ value for the advancing blades and decreased mesh density in the rotor wake.

8 Conclusions

8.1 Summary of Research

This thesis has investigated the development and validation of a novel radial basis function mesh motion method in the context of aerodynamic simulations of helicopter rotors. This new method applies greedy node selection algorithms to the multiscale method of Kedward et al. [2]. This combination grants very high quality mesh deformations, with the exact specification of surface node locations and a near-optimal selection of base nodes. Furthermore, the greedy selection of base nodes avoids potential pitfalls in manual base node selection, which can lead to low quality or even nonviable meshes in scenarios difficult for RBF methods. This comes at a low computational cost due to the implementation of iterative methods, meaning that the overall time investment may be similar or less than coding a manual selection of base nodes. Additionally, this thesis explored the use of variable base node support radii in the context of rotor simulations.

RBF mesh motion has seen increased interest in recent years, with these methods not requiring mesh connectivity data, performing much of their calculations in preprocessing, and achieving high quality meshes where the highest quality degradation occurs away from object surfaces. Whilst the multiscale RBF method of Kedward et al. [2] produced high-quality deformed meshes when changing the collective angle of a quarter-domain hover mesh, this method was challenged in application to the mesh motion of a rotor in hover. This motivated the development of a greedy multiscale algorithm, which performs near-optimal base node selection instead of relying on experientially motivated base node choices. This avoids the potential for specific applications to expose inadequacies in these point selection methods, which was demonstrated in this thesis. Furthermore, near-optimising the base node selection maximises the ratio of mesh quality to base node number. This is desirable as the base nodes are proportionally more computationally expensive than refinement nodes.

Another application focused development was the use of variable support radii for the base nodes. The use of variable support radii has been seen in mathematical applications of RBF methods, but mesh motion applications most often use a constant base node support radius. In Section 4.2, it was shown that the condition number of the base node to base node influence matrix increases when the support radius-scaled distances are smaller. This is an issue for rotor simulations. The locations of the blade roots and hub must be specified, requiring base nodes, but the support radii must be comparatively large due to the blade spans resulting in sizeable deformations. To combat this issue, the support radii were chosen so that the volumes of influence from the blades did not overlap, and did not reach the hub. This method improves the system conditioning by increasing the scaled distance between base nodes at the blade root, as well as by allowing the removal of all base and refinement nodes from the hub. This is achieved while scaling the support radius with the radial location. The radial location correlates with the magnitude of deformation, which is largely a function of the coning. This correlation of support

radius size and deformation magnitude supports high quality deformation.

While these developments addressed the mesh deformation feasibility and quality, greedy RBF methods are already computationally expensive [2, 3], even without the additional calculations required by the multiscale method. As a result, iterative methods were introduced, which drastically shortened the preprocessing time. For the mathematical objects shared between the multiscale methods and base node only RBF methods, the iterative methods were similar to the ones presented in the work of Zhao [3]. However, due to the variable base node support radii, there was a loss in symmetry for many of these matrices, and the techniques needed to be correspondingly adjusted. Iterative methods were also introduced for many of the objects unique to the multiscale methods.

One further development was a new error criterion for the greedy multiscale algorithm. As greedy algorithms are typically used with base node only RBF methods, the surfaces of the moving objects are not fully specified. As such, the error in surface position is the typical error criterion. In contrast, the multiscale method specifies the location of the whole surface, so there is no surface error that could be used as a criterion. After specifying the correct surface geometry, the next characteristic to be desired from the deformation is that it is as close as possible to rigid motion next to the surface. Correspondingly, the new error criterion was defined as the difference between the rigid motion and RBF motion locations of the first layer of volume cells. As points further away from a base node inherently move less than points closer to that node, the distances between the first layer volume nodes and the surface are uniformised.

The effectiveness of the multiscale RBF methods and The University of Sydney flow solver Flamenco were displayed in the hover simulations of the Attack Reconnaissance Class helicopter rotor. The mesh motion techniques were used to change the collective angle for the mesh to 4°, 10° and 12° from a base mesh of 8° collective angle. An analysis of the mesh within a chord length of the blade surface showed very little changes in orthogonality across these meshes. The differences of maximum local orthogonality and mean orthogonality compared to the base mesh are less than 0.5% across the range of collective angles. Furthermore, the largest changes in orthogonality are generally seen at the outer sections of the near blade mesh, which is best for aerodynamics simulations. An analysis of the aerodynamic predictions was also performed, with comparisons to experimental data and other flow solvers. The simulations performed with the multiscale mesh motion and Flamenco solver were seen to be comparably accurate to the other blade-resolved solvers, with similar error in the integrated and sectional load predictions. This accuracy also supported the use of a wall function in rotor aerodynamics simulations. Flamenco employed a wall function with a mesh of target $y^+ = 100$. To the author's knowledge, there are no published rotor simulations using wall functions at such high y^+ values.

With this groundwork, the iterative greedy multiscale RBF method was applied to the forward flight simulation of the Attack Reconnaissance Class helicopter rotor.

Firstly, the preprocessing times for this method were compared to a noniterative implementation to demonstrate their effectiveness. The iterative method proved much faster than the noniterative methods, with the selection of 1000 base nodes taking just over three hours on 256 cores for a mesh with 69 744 surface nodes. This was 80% quicker than the noniterative methods. The iterative method in total also showed superior scaling with the number of base nodes selected and with the number of surface nodes, showing that this advantage in execution speed is a general feature of the algorithms and not limited to the specific system sizes studied.

The mesh quality of the iterative greedy multiscale method was then tested against two other methodologies: the standard multiscale method as presented by Kedward et al. [2] and a multiscale method employing the variable base node support radii. As the base node selection is unspecified for these two approaches, the standard multiscale method employed a selection technique which had been used successfully at The University of Sydney. The variable support radii method selected a set of surface nodes with maximal separation distance, which more evenly distributes the mesh deformation influences and improves the conditioning of the systems of equations. When analysing the deformed meshes, it could be seen that the quality of the mesh generated by the iterative greedy multiscale method was superior to the other two methods. In fact, there were observed sections of the other two meshes with unacceptable mesh geometry. Considering that the iterative greedy multiscale algorithm has no unique mechanism for the actual mesh deformation, this shows the potentially large difficulties in the critical process of base node selection. While a base node selection similar to a greedy algorithm could be achieved, this would require extensive experience, and the time to code such a specific node selection. This makes the iterative greedy multiscale method a preferable algorithm, where the only input required is the number of base nodes to be selected. The guarantee of a near-optimal selection with low computing cost, especially in comparison to the total simulation runtime, is a significant advantage. The aerodynamic predictions of the forward flight calculations were found to be inaccurate in comparison to the other solvers. Considering the mesh deformation of the iterative greedy algorithm was proven to be of high quality, this was likely due to either the wall function becoming nonviable under the simulation conditions, insufficient mesh density, or a combination of these factors.

8.2 Future Work

Whilst the simulations performed in this thesis used the maximum pitching angle as the deformation case for the greedy base node selection, it is not clear if this provides the best grid quality over the full rotation of the rotor. This question of grid quality for time-varying motion can be generalised further, with the desire to produce an optimal base node selection for grid quality over all of the motion maxima and minima. Thus, an investigation into greedy point selection algorithms for such time-dependant deformations may be fruitful. There are a number of potential options for trying to find an optimal base node selection for a time dependant

movement, especially if the movement is not just a scaling of the deformation at one point in time. Intuitively, there is a desire to optimise the base node selection for the points of maximum deformation. Using the helicopter rotor as an example, one would want high mesh quality at the maximum and minimum pitch angle, as well as at the maximum and minimum flap angles. It is not clear if the base node selections for scaled motions, such as maximum and minimum pitch, would even result in the same base node selection, and it is unlikely that these selections would be optimal for flapping deformation. One could conceive of a number of ways to try and optimise the selection over a number of deformation cases, including alternating between maximal deformation cases across the greedy iterations, alternating cases in proportion to the magnitude of the deflections, and alternating between deformation cases corresponding to the components of the motion. There is also the possibility of evaluating the error criterion over multiple deformation cases at each iteration of the greedy algorithm. As demonstrated in Section 4.5, approximately 80% of the iteration time is spent calculating the various influence matrices and running the surface preprocessor. The values of these are independent of deformation, and hence calculating the volume node positions for multiple scenarios would be relatively inexpensive.

While this thesis displays the application of the iterative greedy multiscale algorithm to prespecified mesh motion, the application of this method to Fluid Structure Interaction simulations is the next logical test case. However, this application also raises questions about how to determine the deformed state for point selection. The exact range of structural motion may not be known before simulation, especially if there are no experiments or other simulations of the same scenario. Investigating techniques to produce representative deformed states would be a useful area of research. This could include analysis of simplified structural models with representative applied forces and conservative worst-case estimates of maximum deflections.

Finally, there are aspects of the iterative greedy algorithm that could benefit from further computational improvements. As presented in Section 4.5, the iterative method for $\boldsymbol{\Phi}_{\boldsymbol{r}}$ showed no improvement over the noniterative method, with the cost of copying data masking the speed-ups due to fewer RBF evaluations. Additionally, the surface preprocessor was consistently responsible for over 50% of the time per iteration. With the complexity of this algorithm, there is definitely an opportunity for further optimisation of the iterative methods. One simple avenue would be the parallelisation of these methods, which by itself could drastically cut the runtime.

References

[1] DG Flugzeugbau GmbH, "Dg-300 flattertest / dg-300 flutter testing," Nov. 2016.

[2] L. Kedward, C. B. Allen, and T. Rendall, "Efficient and exact mesh deformation using multi-scale RBF interpolation," *55th AIAA Aerospace Sciences Meeting*, jan 2017.

[3] R. Zhao, C. Li, X. Guo, S. Fan, Y. Wang, and C. Yang, "A block iteration with parallelization method for the greedy selection in radial basis functions based mesh deformation," *Applied Sciences*, vol. 9, p. 1141, Mar. 2019.

[4] Jane's All the World's Aircraft, "Boeing AH-64 Apache," Nov. 2020.

[5] R. Widjaja, J. W. Lim, R. Jain, and M. Potsdam, "Investigation of berp-shape tip design on an apache rotor blade," in *Vertical Flight Society's 76th Annual Forum & Technology Display*, May 2020.

[6] H. Hu, "Study of the trac airfoil table computational system," tech. rep., National Aeronautics and Space Administration, 1999. NASA / CR-1999-209323.

[7] L. S. Stivers, Jr, "Effects of subsonic mach number on the forces and pressure distributions on four naca 64a-series airfoil sections at angles of attack as high as 28°," tech. rep., National Advisory Committe for Aeronautics, 1954.

[8] J. Seddon and S. Newman, *BASIC HELICOPTER AERODYNAMICS*. John Wiley & Sons Ltd, 2011.

[9] W. Johnson, *Rotorcraft Aeromechanics*. Caimbridge University Press, 2013.

[10] M. Benedict, J. Winslow, Z. Hasnain, and I. Chopra, "Experimental Investigation of Micro Air Vehicle Scale Helicopter Rotor in Hover," *International Journal of Micro Air Vehicles*, vol. 7, no. 3, pp. 231–256, 2015.

[11] R. Narducci and H. Tadghighi, "An assessment of create-av helios for apache hover and forward flight simulations," in *54th AIAA Aerospace Sciences Meeting*, 2016.

[12] R. Janakiram, R. Smith, B. Charles, and A. Hassan, "Aerodynamic design of a new affordable main rotor for the apache helicopter," in *American Helicopter Society 59th Annual Forum*, May 2003.

[13] J. D. Kocurek, L. F. Berkowitz, and F. D. Harris, "Hover Performance Methodology at Bell Helicopter Textron," in *American Helicopter Society 36th Annual Forum*, May 1980.

[14] T. R. Quackenbush, D. B. Bliss, D. A. Wachspress, and C. C. Ong, "Free Wake Analysis of Hover Performance Using a New Influence Coefficient Method," tech. rep., NASA Contractor Report 4309, 1990.

[15] T. A. Fitzgibbon, M. A. Woodgate, and G. N. Barakos, "Rotor-Blade Planform Design Based on an Overset Harmonic-Balance-Adjoint Optimization Framework," *AIAA Journal*, vol. 59, no. 9, pp. 3431–3447, 2021.

[16] G. Barakos, "CFD results using HMB3 for an approximate APACHE A blade." Private communication.

[17] J. W. Lim, R. K. Jain, M. A. Potsdam, T. A. Fitzgibbon, G. N. Barakos, and R. Widjaja, "High fidelity code-to-code comparison of rotor performance in hover and forward flight," in *VFS 76th Annual Forum & Technology Display*, Oct. 2020.

[18] S. Osher and S. Chakravarthy, "Upwind schemes and boundary conditions with applications to Euler equations in general geometries," *Journal of Computational Physics*, vol. 50, no. 3, pp. 447–481, 1983.

[19] B. van Leer, "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's Method," *Journal of Computational Physics*, vol. 32, no. 1, pp. 101–136, 1979.

[20] G. D. van Albada, B. van Leer, and W. W. Roberts, "Comparative Study of Computational Methods in Cosmic Gas Dynamics," *Astronomy and Astrophysics*, vol. 108, no. 1, pp. 76–84, 108.

[21] J. L. Steger, F. C. Dougherty, and J. A. Benek, "A chimera grid scheme," *Advances in grid generation; Proceedings of the Applied Mechanics, Bioengineering, and Fluids Engineering Conference*, pp. 59–69, June 1983.

[22] T. Schwarz, F. Spiering, and N. Kroll, "Grid coupling by means of chimera interpolation techniques," in *Second Symposium "Simulation of Wing and Nacelle Stall"*, June 2010.

[23] J. Sitaraman, M. Floros, A. Wissink, and M. Potsdam, "Parallel domain connectivity algorithm for unsteady flow computations using overlapping and adaptive grids," *Journal of Computational Physics*, vol. 229, pp. 4703–4723, June 2010.

[24] Z. WANG, N. HARIHARAN, and R. CHEN, "Recent development on the conservation property of chimera," *International Journal of Computational Fluid Dynamics*, vol. 15, pp. 265–278, Nov. 2001.

[25] R. Meakin, "On the spatial and temporal accuracy of overset grid methods for moving body problems," in *12th Applied Aerodynamics Conference*, American Institute of Aeronautics and Astronautics, June 1994.

[26] Z. Wang, "A fully conservative interface algorithm for overlapped grids," *Journal of Computational Physics*, vol. 122, pp. 96–106, Nov. 1995.

[27] D. DAVIS and H. THOMPSON, "The impact of computational zone interfacing on calculated scramjet performance," in *30th Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics, Jan. 1992.

[28] J. Liu and W. Shyy, "Assessment of grid interface treatments for multi-block incompressible viscous flow computation," *Computers & Fluids*, vol. 25, pp. 719–740, Nov. 1996.

[29] M. Mohan Rai, "A conservative treatment of zonal boundaries for euler equation calculations," *Journal of Computational Physics*, vol. 62, pp. 472–503, Feb. 1986.

[30] C. S. Peskin, "The fluid dynamics of heart valves: Experimental, theoretical, and computational methods," *Annual Review of Fluid Mechanics*, vol. 14, pp. 235–259, Jan. 1982.

[31] E. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof, "Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations," *Journal of Computational Physics*, vol. 161, pp. 35–60, June 2000.

[32] J. Yang and E. Balaras, "An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries," *Journal of Computational Physics*, vol. 215, pp. 12–40, June 2006.

[33] R. Mittal and G. Iaccarino, "Immersed boundary methods," *Annual Review of Fluid Mechanics*, vol. 37, pp. 239–261, Jan. 2005.

[34] F. Capizzano and D. Cinquegrana, "Applying a cartesian method to moving boundaries," *Computers & Fluids*, vol. 263, p. 105968, Sept. 2023.

[35] R. Verzicco, "Immersed boundary methods: Historical perspective and future outlook," *Annual Review of Fluid Mechanics*, vol. 55, pp. 129–155, Jan. 2023.

[36] J. Lee, J. Kim, H. Choi, and K.-S. Yang, "Sources of spurious force oscillations from an immersed boundary method for moving-body problems," *Journal of Computational Physics*, vol. 230, pp. 2677–2695, Apr. 2011.

[37] J. H. Seo and R. Mittal, "A sharp-interface immersed boundary method with improved mass conservation and reduced spurious pressure oscillations," *Journal of Computational Physics*, vol. 230, pp. 7347–7363, Aug. 2011.

[38] A. M. Roma, C. S. Peskin, and M. J. Berger, "An adaptive version of the immersed boundary method," *Journal of Computational Physics*, vol. 153, pp. 509–534, Aug. 1999.

[39] D. Rettenmaier, D. Deising, Y. Ouedraogo, E. Gjonaj, H. De Gersem, D. Bothe, C. Tropea, and H. Marschall, "Load balanced 2d and 3d adaptive mesh refinement in openfoam," *SoftwareX*, vol. 10, p. 100317, July 2019.

[40] W. J. Gordon and C. A. Hall, "Construction of curvilinear co-ordinate systems and applications to mesh generation," *International Journal for Numerical Methods in Engineering*, vol. 7, pp. 461–477, Jan. 1973.

[41] C. B. Allen, "Central-difference and upwind-biased schemes for steady and unsteady euler aerofoil computations," *The Aeronautical Journal*, vol. 99, pp. 52–62, Feb. 1995.

[42] C. B. Allen, "The reduction of numerical entropy generated by unsteady shockwaves," *The Aeronautical Journal*, vol. 101, pp. 9–16, Jan. 1997.

[43] C. B. Allen, "Grid adaptation for unsteady flow computations," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 211, pp. 237–250, Apr. 1997.

[44] A. L. Gaitonde, "A dual-time method for the solution of the unsteady euler equations," *The Aeronautical Journal*, vol. 98, pp. 283–291, Oct. 1994.

[45] A. L. Gaitonde and S. P. Fiddes, "A comparison of a cell-centre method and a cell-vertex method for the solution of the two-dimensional unsteady euler equations on a moving grid," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 209, pp. 203–213, July 1995.

[46] C. B. Allen, "Aeroelastic computations using algebraic grid motion," *The Aeronautical Journal*, vol. 106, pp. 559–570, Oct. 2002.

[47] L. Dubuc, M. Cantariti, M. Woodgate, B. Gribben, K. Badcock, and B. Richards, "Solution of the Euler Unsteady Equations Using Deforming Grids," tech. rep., University of Glasgow, 1997.

[48] K. Badcock, S. Timme, S. Marques, H. Khodaparast, M. Prandina, J. Mottershead, A. Swift, A. Da Ronch, and M. Woodgate, "Transonic aeroelastic simulation for instability searches and uncertainty analysis," *Progress in Aerospace Sciences*, vol. 47, pp. 392–423, July 2011.

[49] M. Carrión, R. Steijl, M. Woodgate, G. Barakos, X. Munduate, and S. Gomez-Iradi, "Aeroelastic analysis of wind turbines using a tightly coupled cfd–csd method," *Journal of Fluids and Structures*, vol. 50, pp. 392–415, Oct. 2014.

[50] J. T. Batina, "Unsteady euler algorithm with unstructured dynamic mesh for complex-aircraft aerodynamic analysis," *AIAA Journal*, vol. 29, pp. 327–333, Mar. 1991.

[51] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne, "Torsional springs for two-dimensional dynamic unstructured fluid meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 163, pp. 231–245, Sept. 1998.

[52] K. P. Singh, J. C. Newman, and O. Baysal, "Dynamic unstructured method for flows past multiple objects in relative motion," *AIAA Journal*, vol. 33, pp. 641–649, Apr. 1995.

[53] S. Piperno, "Explicit/implicit fluid/structure staggered procedures with a structural predictor and fluid subcycling for 2d inviscid aeroelastic simulations," *International Journal for Numerical Methods in Fluids*, vol. 25, pp. 1207–1226, Nov. 1997.

[54] F. J. Blom, "Considerations on the spring analogy," *International Journal for Numerical Methods in Fluids*, vol. 32, pp. 647–668, Mar. 2000.

[55] C. Degand and C. Farhat, "A three-dimensional torsional spring analogy method for unstructured dynamic meshes," *Computers & Structures*, vol. 80, pp. 305–316, Feb. 2002.

[56] P. Geuzaine, G. Brown, C. Harris, and C. Farhat, "Aeroelastic dynamic analysis of a full f-16 configuration for various flight conditions," *AIAA Journal*, vol. 41, pp. 363–371, Mar. 2003.

[57] T. Lieu and C. Farhat, "Adaptation of aeroelastic reduced-order models and application to an f-16 configuration," *AIAA Journal*, vol. 45, pp. 1244–1257, June 2007.

[58] R. Löhner and C. Yang, "Improved ale mesh velocities for moving bodies," *Communications in Numerical Methods in Engineering*, vol. 12, pp. 599–608, Oct. 1996.

[59] F. M. Bos, B. W. van Oudheusden, and H. Bijl, "Radial basis function based mesh deformation applied to simulation of flow around flapping wings," *Computers & Fluids*, vol. 79, pp. 167–177, June 2013.

[60] B. T. Helenbrook, "Mesh deformation using the biharmonic operator," *International Journal for Numerical Methods in Engineering*, vol. 56, pp. 1007–1021, Jan. 2003.

[61] J. Baum, H. Luo, R. Loehner, E. Goldberg, A. Feldhun, J. Baum, H. Luo, R. Loehner, E. Goldberg, and A. Feldhun, "Application of unstructured adaptive moving body methodology to the simulation of fuel tank separation from an f-16 fighter," in *35th Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics, Jan. 1997.

[62] L. Dempere-Marco, E. Oubel, M. Castro, C. Putman, A. Frangi, and J. Cebral, *CFD Analysis Incorporating the Influence of Wall Motion: Application to Intracranial Aneurysms*, pp. 438–445. Springer Berlin Heidelberg, 2006.

[63] T. C. S. Rendall and C. B. Allen, "Unified fluid-structure interpolation and mesh motion using radial basis functions," *International Journal for Numerical Methods in Engineering*, vol. 74, pp. 1519–1559, 2008.

[64] X. Liu, N. Qin, and H. Xia, "Fast dynamic grid deformation based on delaunay graph mapping," *Journal of Computational Physics*, vol. 211, pp. 405–423, Jan. 2006.

[65] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, and D. G. Kendall, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Inc., July 2000.

[66] Y. Wang, N. Qin, and N. Zhao, "Delaunay graph and radial basis function for fast quality mesh deformation," *Journal of Computational Physics*, vol. 294, pp. 149–172, Aug. 2015.

[67] A. de Boer, M. S. van der Schoot, and H. Bijl, "Mesh deformation based on radial basis function interpolation," *Computers & Structures*, vol. 85, pp. 784–795, 2007.

[68] T. C. S. Rendall and C. B. Allen, "Efficient mesh motion using radial basis functions with datareduction algorithms," *Journal of Computational Physics*, 2009.

[69] H. Wendland, *Konstruktion und Untersuchung radialer Basisfunktionen mit kompaktem Träger*. PhD thesis, Georg-August-Universität zu Göttingen, 1996.

[70] H. Wendland, *Scattered Data Approximation*. Cambridge University Press, Dec. 2004.

[71] M. S. Floater and A. Iske, "Multistep scattered data interpolation using compactly supported radial basis functions," *Journal of Computational and Applied Mathematics*, vol. 73, pp. 65–78, Oct. 1996.

[72] A. Murray, *Radial Basis Function Methods in Fluid-Structure Interaction*. PhD thesis, The University of Sydney, 2023.

[73] N. Flyer, B. Fornberg, V. Bayona, and G. A. Barnett, "On the role of polynomials in rbf-fd approximations: I. interpolation and accuracy," *Journal of Computational Physics*, vol. 321, pp. 21–38, Sept. 2016.

[74] V. Bayona, N. Flyer, B. Fornberg, and G. A. Barnett, "On the role of polynomials in rbf-fd approximations: Ii. numerical solution of elliptic pdes," *Journal of Computational Physics*, vol. 332, pp. 257–273, Mar. 2017.

[75] V. Bayona, N. Flyer, and B. Fornberg, "On the role of polynomials in rbf-fd approximations: Iii. behavior near domain boundaries," *Journal of Computational Physics*, vol. 380, pp. 378–399, Mar. 2019.

[76] T. C. S. Rendall and C. B. Allen, "Parallel efficient mesh motion using radial basis functions with application to multi-bladed rotors," *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, vol. 81, pp. 89–105, 2010.

[77] T. C. S. Rendall and C. B. Allen, "Reduced surface point selection options for efficient mesh deformationusing radial basis functions," *Journal of Computational Physics*, vol. 229, pp. 2810–2820, Dec. 2009.

[78] O. Estruch, O. Lehmkuhl, R. Borrell, C. Pérez Segarra, and A. Oliva, "A parallel radial basis function interpolation method for unstructured dynamic meshes," *Computers & Fluids*, vol. 80, pp. 44–54, July 2013.

[79] A. Beckert and H. Wendland, "Multivariate interpolation for fluid-structure-interaction problems using radial basis functions," *Aerospace Science and Technology*, vol. 5, pp. 125–134, Feb. 2001.

[80] H. JIA and Q. SUN, "A comparison of two dynamic mesh methods in fluid –structure interaction," in *Proceedings of the 2nd International Conference on Electronic and Mechanical Engineering and Information Technology (2012)*, EMEIT-12, Atlantis Press, 2012.

[81] Y. Rozenberg, S. Aubert, and G. Bénéfice, "Fluid structure interaction problems in turbomachinery using rbf interpolation and greedy algorithm," in *Volume 2B: Turbomachinery*, GT2014, American Society of Mechanical Engineers, June 2014.

[82] S. Jakobsson and O. Amoignon, "Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization," *Computers & Fluids*, vol. 36, pp. 1119–1136, July 2007.

[83] R. Schaback and H. Wendland, "Adaptive greedy techniques for approximate solution of large rbf systems," *Numerical Algorithms*, vol. 24, no. 3, pp. 239–254, 2000.

[84] R. Schaback and H. Wendland, "Numerical techniques based on radial basis functions," in *Curve and Surface Fitting: Saint-Malo 99* (A. Cohen, C. Rabut, and L. L. Schumaker, eds.), Vanderbilt University Press, 2000.

[85] S. De Marchi, "On optimal center locations for radial basis function interpolation: Computational aspects," *Rendiconti Del Seminario Matematico Università e Politecnico di Torino*, vol. 61, no. 3, pp. 343–358, 2003.

[86] C. Li, X. Xu, J. Wang, L. Xu, S. Ye, and X. Yang, "A parallel multiselection greedy method for the radial basis function–based mesh deformation," *International Journal for Numerical Methods in Engineering*, vol. 113, pp. 1561–1588, Nov. 2017.

[87] S. Kern and P. Koumoutsakos, "Simulations of optimized anguilliform swimming," *Journal of Experimental Biology*, vol. 209, pp. 4841–4857, Dec. 2006.

[88] T.-T. Lu and S.-H. Shiou, "Inverses of 2 × 2 block matrices," *Computers & Mathematics with Applications*, vol. 43, pp. 119–129, Jan. 2002.

[89] V. Skala, "Radial basis functions interpolation and applications: An incremental approach," in *ASM 2010 Conference*, 2010.

[90] O. Amoignon and M. Berggren, "Adjoint of a median-dual finite-volume scheme application to transonic aerodynamic shape optimization," tech. rep.,

Department of Information Technology, Uppsala University, Uppsala, Sweden, 2006.

[91] M. M. Selim and R. Koomullil, "Mesh deformation approaches – a survey," *Journal of Physical Mathematics*, vol. 7, no. 2, 2016.

[92] A. Murray, B. Thornber, M. Flaig, and G. Vio, "Highly parallel, multi-stage mesh motion using radial basis functions for fluid-structure interaction," in *AIAA Scitech 2019 Forum*, 2019.

[93] M. Sipser, *Introduction to the theory of computation*. Australia: Cengage Learning, third edition, international edition ed., 2013. Hier auch später erschienene, unveränderte Nachdrucke.

[94] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press Series, MIT Press, 3 ed., 2009.

[95] D. Linton, G. Barakos, R. Widjaja, and B. Thornber, "New actuator surface model with improved wake model for cfd simulationsof rotorcraft," *73rd Annual Forum and Technology Display of the American Helicopter Society*, 2017.

[96] D. Linton, R. Widjaja, and B. Thornber, "Validation of an Actuator Surface Model with CFD-convected Wake Model for Hover and Forward Flight," in *7th Asian/Australian Rotorcraft Forum*, 2018.

[97] D. Linton, R. Widjaja, and B. Thornber, "Simulations of Tandem and Coaxial Rotors using a CFD-coupled Rotor Model," in *21st Australasian Fluid Mechanics Conference*, (Adelaide, Australia), Dec. 2018.

[98] D. Linton, G. Barakos, R. Widjaja, and B. Thornber, "Coupling of an Unsteady Aerodynamics Model with a Computational Fluid Dynamics Solver," *AIAA Journal*, vol. 56, Aug. 2018.

[99] D. Linton and B. Thornber, "An actuator surface method for ship-helicopter dynamic interface simulations," American Institute of Aeronautics and Astronautics, jan 2021.

[100] D. Linton, *A Hybrid Computational Fluid Dynamics Method for Unsteady Simulation of the Ship-Helicopter Dynamic Interface*. PhD thesis, The University Of Sydney, Dec. 2019.

[101] D. Linton, *ASM User Guide*, 1 ed., Mar. 2021.

[102] T. Beddoes, "Practical computation of unsteady lift," *Vertica*, vol. 8, no. 1, 1984.

[103] T. Kim, S. Oh, and K. Yee, "Improved actuator surface method for wind turbine application," *Renewable Energy*, vol. 76, pp. 16–26, Apr. 2015.

[104] M. Ramasamy and J. Leishman, "A reynolds number-based blade tip vortex model," *Journal of the American Helicopter Society*, vol. 52, no. 3, pp. 214–223, 2007.

[105] A. Garcia-Uceda Juarez, A. Raimo, E. Shapiro, and B. Thornber, "Steady turbulent flow computations using a low mach fully compressible scheme," *AIAA Journal*, vol. 52, no. 1, pp. 2559–2575, 2014.

[106] P. D. Thomas and C. K. Lombard, "Geometric conservation law and its application to flow computations on moving grids," *AIAA Journal*, vol. 17, pp. 1030–1037, Oct. 1979.

[107] T. H. Pulliam and J. L. Steger, "Implicit finite-difference simulations of three-dimensional compressible flow," *AIAA Journal*, vol. 18, pp. 159–167, Feb. 1980.

[108] E. F. Toro, M. Spruce, and W. Speares, "Restoration of the contact surface in the hll-riemann solver," *Shock Waves*, vol. 4, pp. 25–34, July 1994.

[109] A. Bagabir and D. Drikakis, "Numerical experiments using high-resolution schemes for unsteady, inviscid, compressible flows," vol. 193, pp. 4675–4705, oct 2004.

[110] K. H. Kim and C. Kim, "Accurate, efficient and monotonic numerical methods for multi-dimensional compressible flows Part I: Spatial discretization," *Journal of Computational Physics*, vol. 208, pp. 527–567, 2005.

[111] R. J. Spiteri and S. J. Ruuth, "A new class of optimal high-order strong-stability-preserving time discretization methods," *SIAM Journal on Numerical Analysis*, vol. 40, pp. 469–491, Jan. 2002.

[112] B. Thornber, A. Mosedale, D. Drikakis, and D. Youngs, "An improved reconstruction method for compressible flows with low Mach number features," *Journal of Computational Physics*, vol. 227, pp. 4873–4894, May 2008.

[113] P. R. Spalart and S. R. Allmaras, "A one-equation turbulence model for aerodynamic flows," in *30th Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics, Jan. 1992.

[114] P. R. Spalart and S. R. Allmaras, "A one-equation turbulence model for aerodynamic flows," in *La Recherche Aérospatiale*, no. 1, pp. 5–21, 1994.

[115] S. R. Allmaras, F. T. Johnson, and P. R. Spalart, "Modifications and clarifications for the implementation of the spalart-allmaras turbulence model," in *Seventh international conference on computational fluid dynamics (ICCFD7)*, vol. 1902, Big Island, HI, 2012.

[116] T. Fitzgibbon, M. Woodgate, and G. Barakos, "Assessment of current rotor design comparison practices based on high-fidelity cfd methods," *The Aeronautical Journal*, vol. 124, pp. 731–766, Jan. 2020.

[117] G. N. Barakos and A. J. Garica, "CFD analysis of hover performance of rotors at full- and model-scale conditions," *The Aeronautical Journal*, vol. 120, pp. 1386–1424, Sept. 2016.

[118] R. Jain, "Cfd performance and turbulence transition predictions on an installed model-scale rotor in hover," in *55th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, Jan. 2017.

[119] A. Bodling and M. Potsdam, "Numerical investigation of secondary vortex structures in a rotorwake," in *VFS International 76th Annual Forum & Technology Display*, Oct. 2020.

[120] W. Sutherland, "The viscosity of gases and molecular force," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 36, pp. 507–531, Dec. 1893.

[121] P. J. Pritchard, *Fox and McDonald's introduction to fluid mechanics*. John Wiley & Sons, 2011.

[122] F. P. Incropera, D. P. DeWitt, T. L. Bergman, and A. S. Lavine, *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons Ltd, 2007.

[123] D. B. Spalding, "A single formula for the "law of the wall"," *Journal of Applied Mechanics*, vol. 28, pp. 455–458, Sept. 1961.

[124] A. Coyle and B. Thornber, "Actuator Surface Model and Blade Resolved Simulations of an Attack Reconnaissance Class Rotor in Hover and Forward Flight," in *49th European Rotorcraft Forum 2023 - Proceedings*, 2023.

[125] A. O. Yüksel, "NAVIER-STOKES COMPUTATIONS OF HELICOPTER ROTOR FLOW FIELDS," Master's thesis, Middle East Technical University, Sept. 2017.

[126] R. Jain, "Hover predictions for the S-76 rotor with tip shape variation using CREATE-AV helios," in *53rd AIAA Aerospace Sciences Meeting*, Jan. 2015.

[127] L. I. Garipova, A. S. Batrakov, A. N. Kusyumov, S. A. Mikhaylov, and G. Barakos, "Aerodynamic and acoustic analysis of helicopter main rotor blade tips in hover," *International Journal of Numerical Methods for Heat and Fluid Flow*, vol. 26, no. 7, pp. 2101–2118, 2016.

[128] C. Johnson, *Optimisation of Aspects of Rotor Blades using Computational Fluid Dynamics*. PhD thesis, The University of Liverpool, June 2012.

[129] M. A. Potsdam and R. C. Strawn, "CFD simulations of tiltrotor configurations in hover," *Journal of the American Helicopter Society*, vol. 50, pp. 82–94, Jan. 2005.

[130] B. Siebert and G. Dulikravich, "Grid generation using a posteriori optimization with geometrically normalized functionals," in *Flight Simulation Technologies Conference and Exhibit*, American Institute of Aeronautics and Astronautics, Aug. 1990.

[131] K. Kallstrom, "Exploring Airfoil Table Generation using XFOIL and OVERFLOW," in *Aeromechanics for Advanced Vertical Flight Technical Meeting, Transformative Vertical Flight*, 2022.

[132] M. Drela and H. Youngren, *XFOIL 6.94 User Guide*, Dec. 2001.

[133] H. McDonald, "The effect of pressure gradient on the law of the wall in turbulent flow," *Journal of Fluid Mechanics*, vol. 35, pp. 311–336, Jan. 1969.

[134] C. Mockett, M. Fuchs, and F. Thiele, "Progress in des for wall-modelled les of complex internal flows," *Computers &; Fluids*, vol. 65, pp. 44–55, July 2012.

[135] W. J. McCroskey, K. W. McAlister, L. W. Carr, and S. L. Pucci, "An experimental study of dynamic stall on advanced airfoil sections volume 1. summary of the experiment," tech. rep., National Aeronautics and Space Administration, 1982.

[136] C. C. Wolf, C. Schwarz, K. Kaufmann, A. D. Gardner, D. Michaelis, J. Bosbach, D. Schanz, and A. Schröder, "Experimental study of secondary vortex structures in a rotor wake," *Experiments in Fluids*, vol. 60, Oct. 2019.

[137] P. R. Spalart, S. Deck, M. L. Shur, K. D. Squires, M. K. Strelets, and A. Travin, "A new version of detached-eddy simulation, resistant to ambiguous grid densities," *Theoretical and Computational Fluid Dynamics*, vol. 20, pp. 181–195, May 2006.

[138] U. Piomelli, "Wall-layer models for large-eddy simulations," *Progress in Aerospace Sciences*, vol. 44, pp. 437–446, Aug. 2008.

[139] P. Morris, J. Ottomeyer, L. Higgins, G. Bender, B. Picasso, III, and R. Savage, "Airworthiness and flight characteristics test part 1 yah-64 attack helicopter," tech. rep., USAAEFA Project No. 80-17-1, 1981.

[140] M. Kadivar, D. Tormey, and G. McGranaghan, "A review on turbulent flow over rough surfaces: Fundamentals and theories," *International Journal of Thermofluids*, vol. 10, p. 100077, May 2021.

[141] D. K. Kolmogorov, F. Menter, and A. V. Garbaruk, "On mesh requirements for large eddy simulation with wall functions," *Journal of Physics: Conference Series*, vol. 2103, p. 012212, Nov. 2021.

[142] T. Craft, A. Gerasimov, H. Iacovides, and B. Launder, "Progress in the generalization of wall-function treatments," *International Journal of Heat and Fluid Flow*, vol. 23, pp. 148–160, Apr. 2002.

A Appendix A - Examples of Φ Condition Contours

This appendix provides more examples of contour plots for the condition of **Φ**, with varying base node locations. The value of the contour at specific coordinates is the condition number of **Φ** with base nodes at the red crosses and one base node at the same coordinates. Contours with uniform and nonuniform base node support radii are shown.

Circular Arrangement, Variable Support Radii

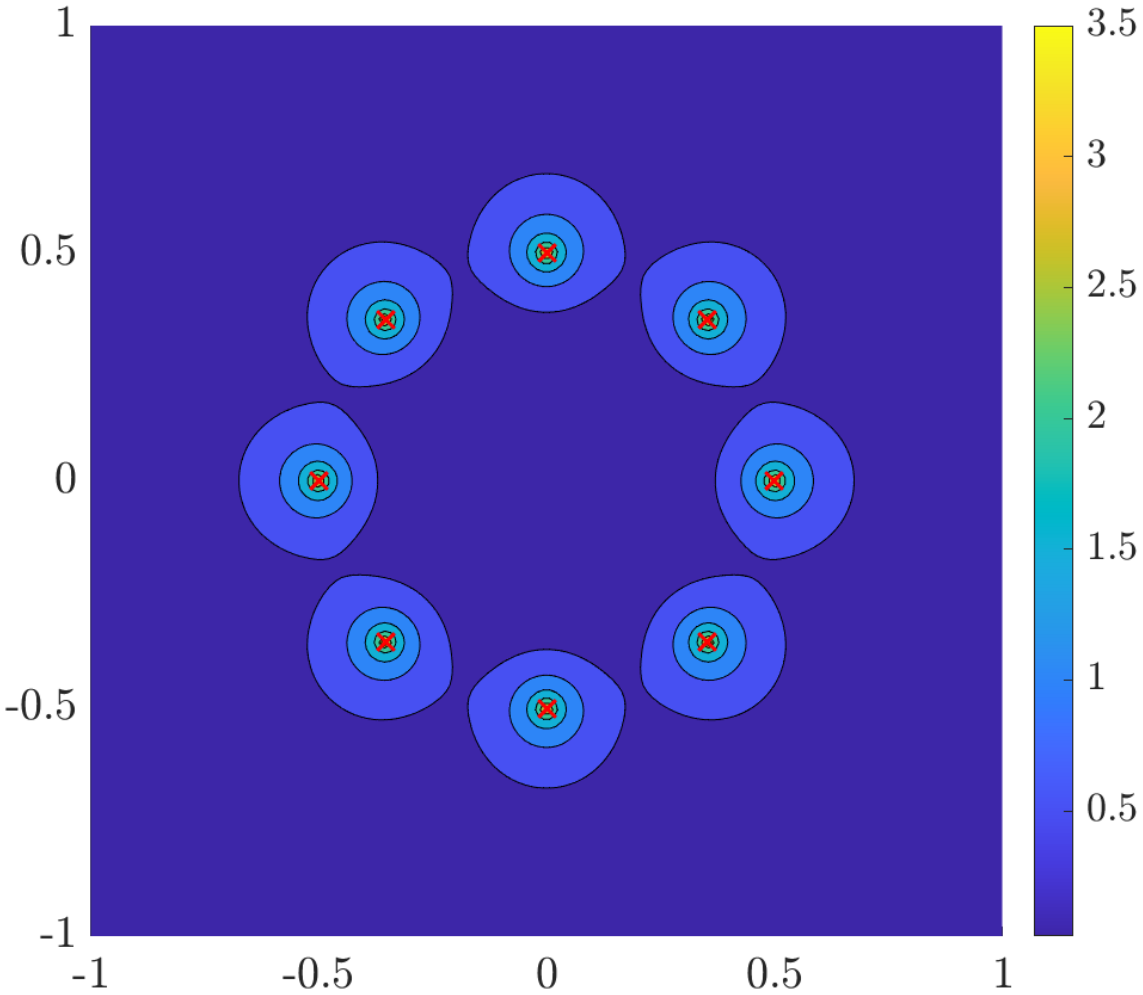


Figure 82: Circle of base nodes, variable support radii

Square Arrangement

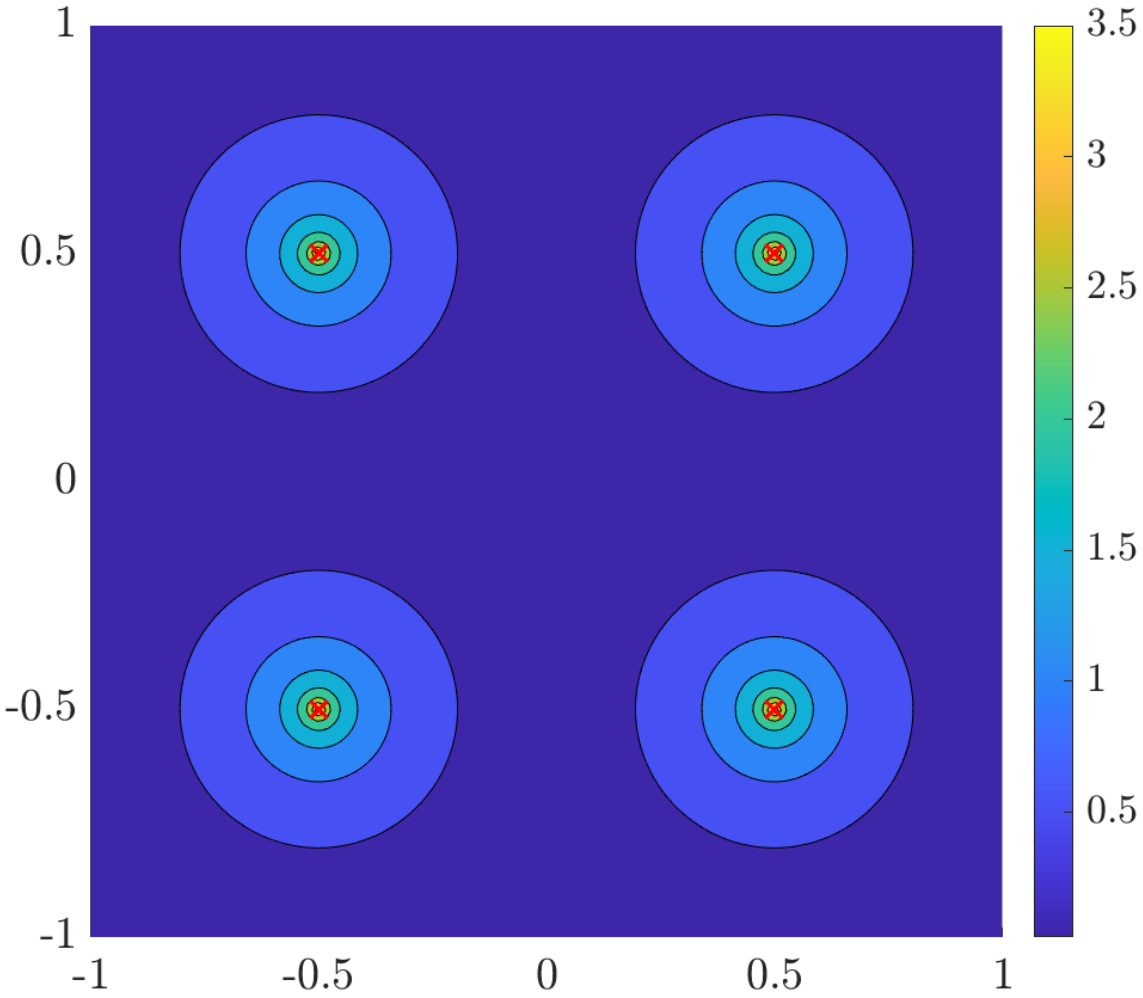


Figure 83: Square of base nodes, constant support radii

Square Arrangement, Variable Support Radii

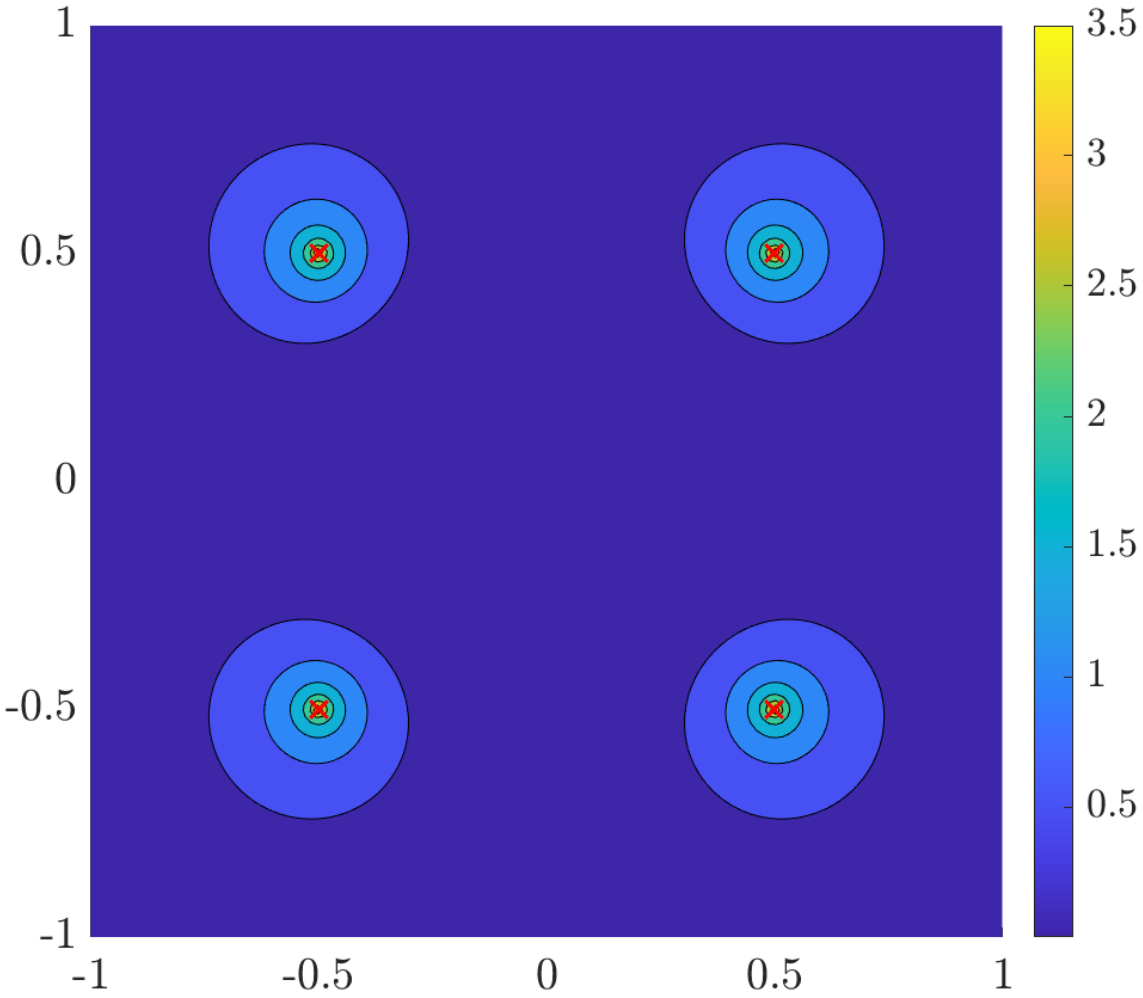


Figure 84: Square of base nodes, variable support radii

Square Arrangement, Off Centre

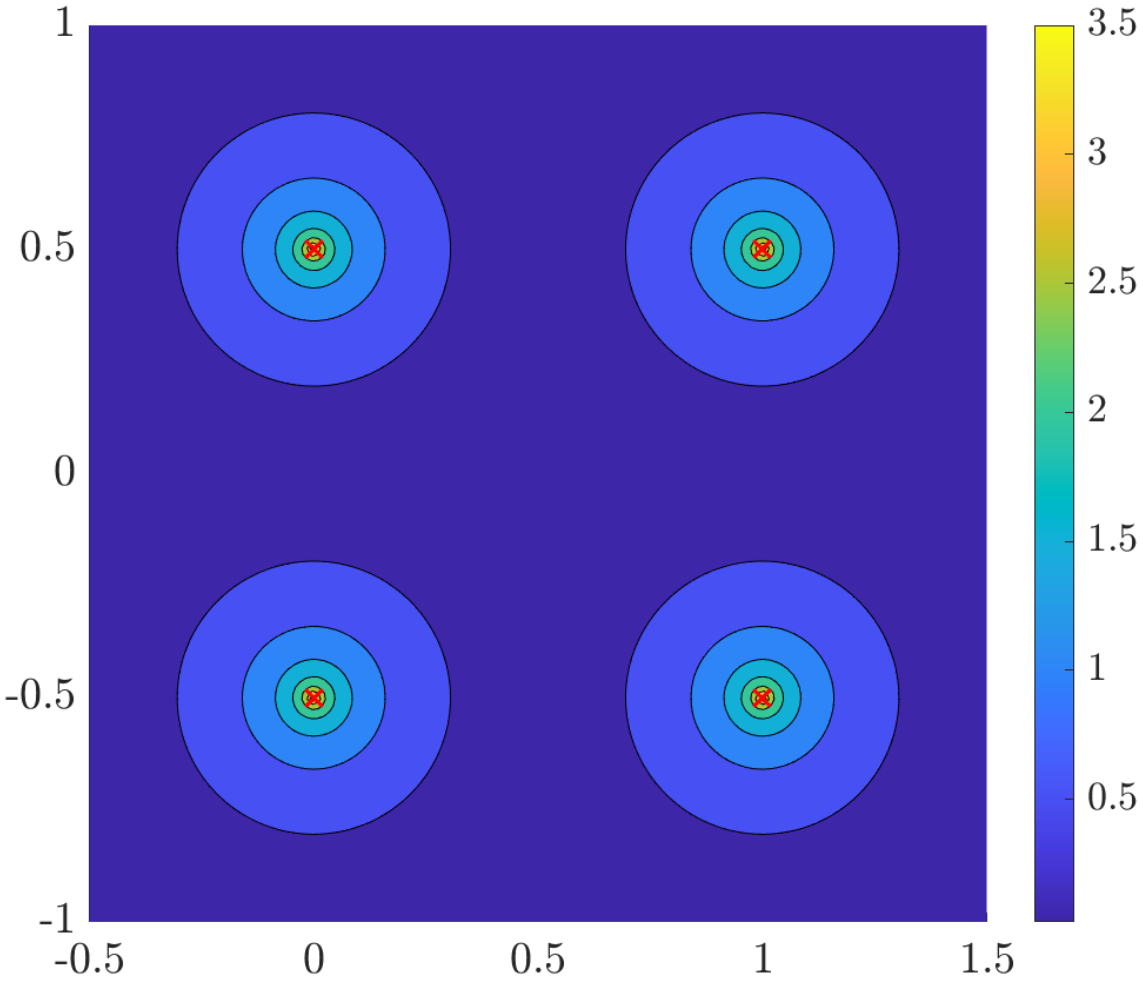


Figure 85: Square of base nodes off centre, constant support radii

Square Arrangement, Off Centre, Variable Support Radii

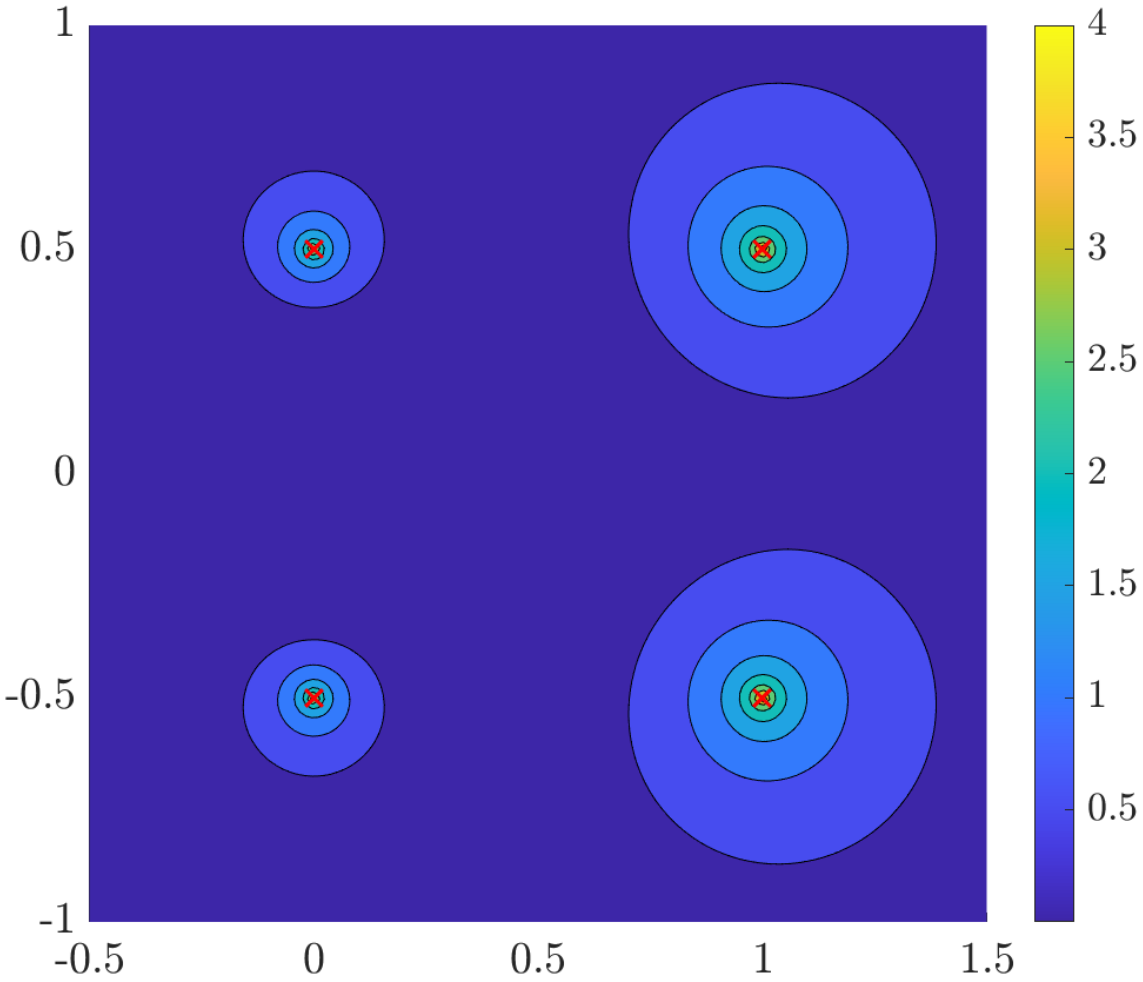


Figure 86: Square of base nodes off centre, variable support radii

Tapered, Swept Wing

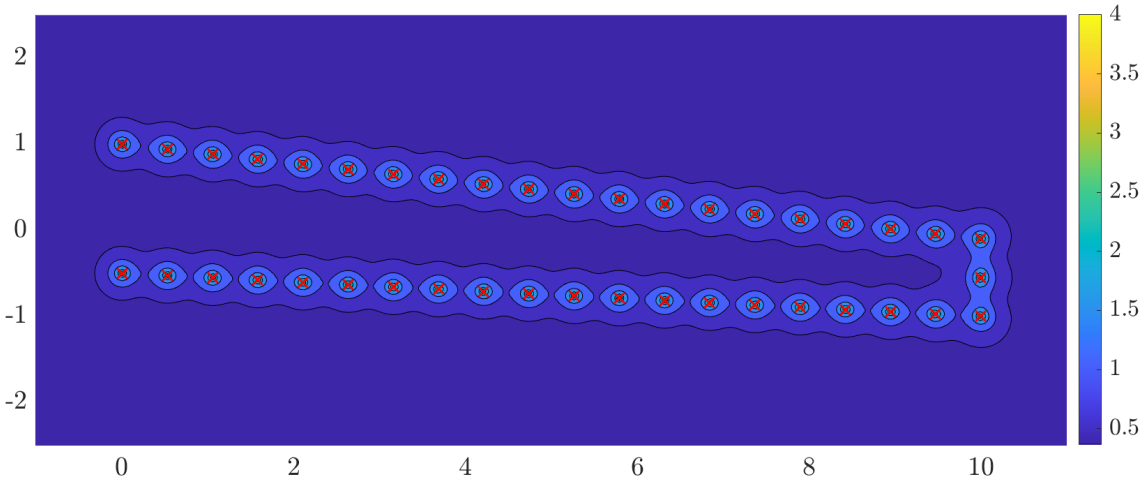


Figure 87: Base nodes in the shape of a tapered, swept wing, constant support radii

Tapered, Swept Wing, Variable Support Radii

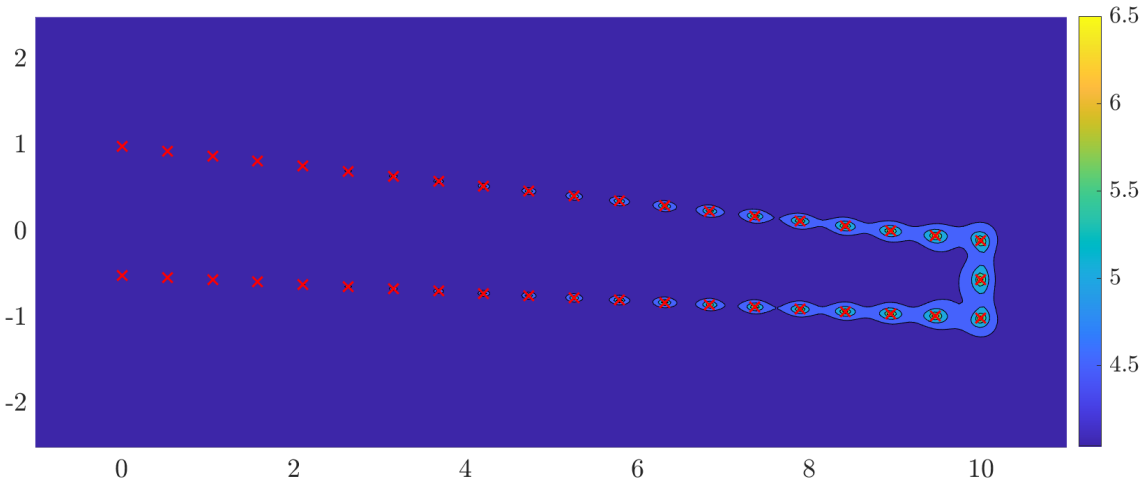
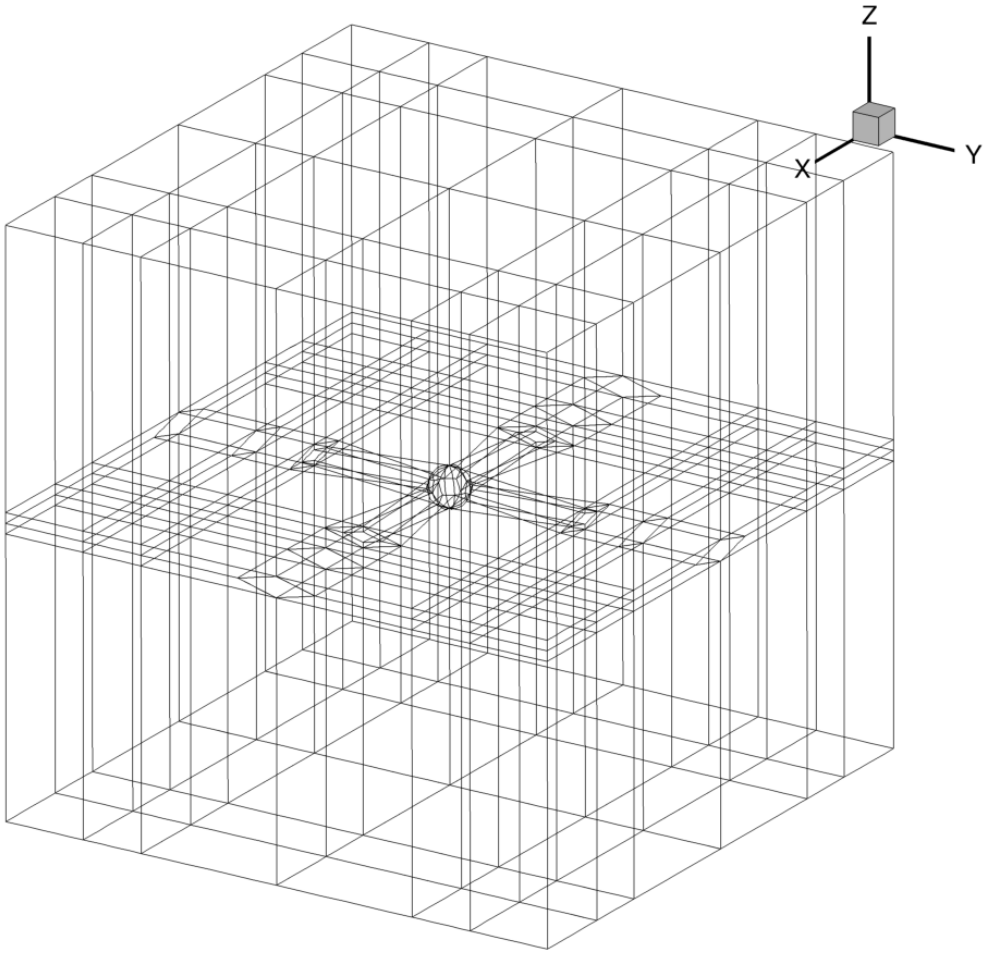


Figure 88: Base nodes in the shape of a tapered, swept wing, variable support radii

B Appendix B - Diagram of Diamond Aerofoil Rotor

The diamond aerofoil rotor used for a comparison case in Section 4.5.1 is shown in the figure below. Each segment seen in the diagram contains 7 mesh nodes.



C Appendix C - Additional Computational Costs

This appendix outlines the asymptotic cost of iterating $\mathbf{L}$ in the worst-case scenario when a $O(n \log n)$ sorting algorithm is used. A number of common sorting algorithms have this asymptotic cost, including Heapsort, Quicksort and Mergesort. When using such a sorting algorithm, the most expensive part of Algorithm 3 is the sorting of each row of $\mathbf{L}_i$. With N_r rows and a maximum of j elements in each row, the total sorting cost is:

$$\sum_{j=1}^{N_r} O\left(j \log j\right) = O\left(\log\left(H(N_r)\right)\right) \tag{174}$$

Where $H(x)$ is the hyperfactorial of x, defined as $H(x) = \prod_{i=1}^{x} i^i$. Equation (174) can be shown to grow faster than $O\left(n^2\right)$ but slower than all functions $O\left(n^\alpha\right), \alpha > 2$. Firstly re-express Equation (174) as the order of the sum:

$$\sum_{j=1}^{N_r} O\left(j \log j\right) = O\left(\sum_{j=1}^{N_r} j \log j\right) \tag{175}$$

The sum of $j \log j$ from 1 to n can then be shown to be less than $n^2 \log n$

$$\sum_{j=1}^{n} j \log j = \sum_{j=1}^{n} \log\left(j^j\right) < \sum_{j=1}^{n} \log\left(n^n\right) = n \log\left(n^n\right) = n^2 \log\left(n\right) \tag{176}$$

Then compare the limit of the ratio of $n^2 \log n$ to a polynomial of arbitrary degree α as n approaches infinity:

$$\lim_{n\to\infty} \frac{n^2 \log(n)}{n^\alpha} = \lim_{n\to\infty} \frac{\log(n)}{n^{\alpha-2}} = \lim_{n\to\infty} \frac{1/n}{(\alpha-2)n^{\alpha-3}} = \lim_{n\to\infty} \frac{1}{\alpha-2} n^{2-\alpha} \tag{177}$$

This final expression approaches 0 for $\alpha > 2$, and grows unbounded for $\alpha < 2$. In total, this means that when using a $O\left(n \log n\right)$ sorting algorithm, as well as encountering the worst scenario of $\mathbf{L}_i$ containing no elements that can be copied from $\mathbf{L}_{i-1}$ and no zero elements, the asymptotic cost is only slightly worse than calculating $\mathbf{L}_i$ from scratch.

D Appendix D - Timing Graphs with Log-Log Axes

This appendix presents the same graphs as presented in Section 4.5, but plotted on logarithmic axes instead of linear. This allows for a visual comparison of the algorithm scaling exponents through comparison of gradients.

Surface Preprocessor

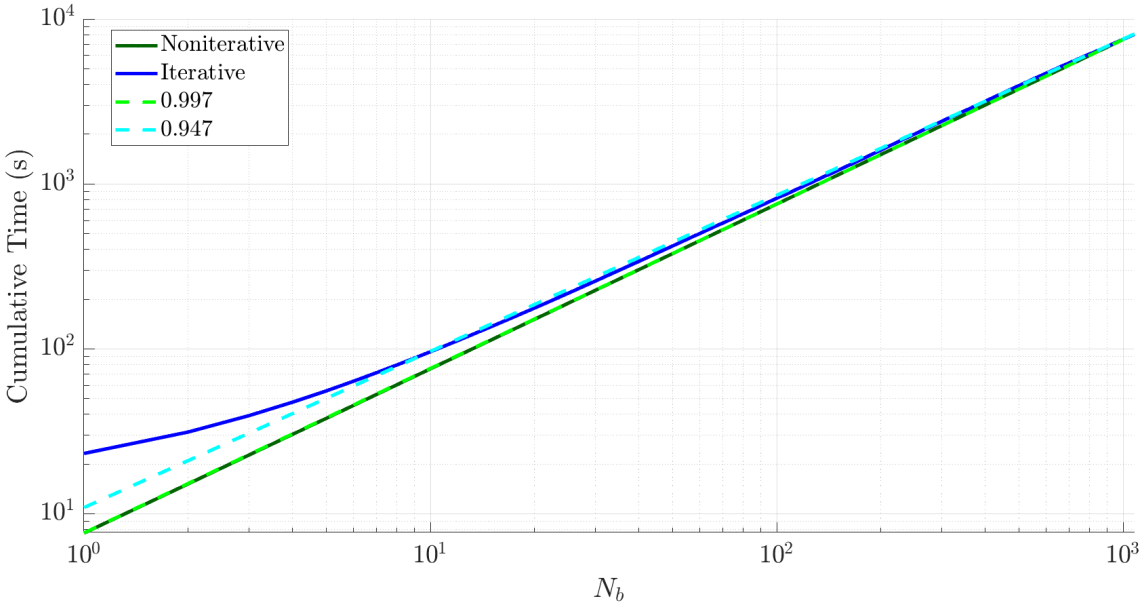


Figure 89: Cumulative time taken by the preprocessor in choosing N_b base nodes

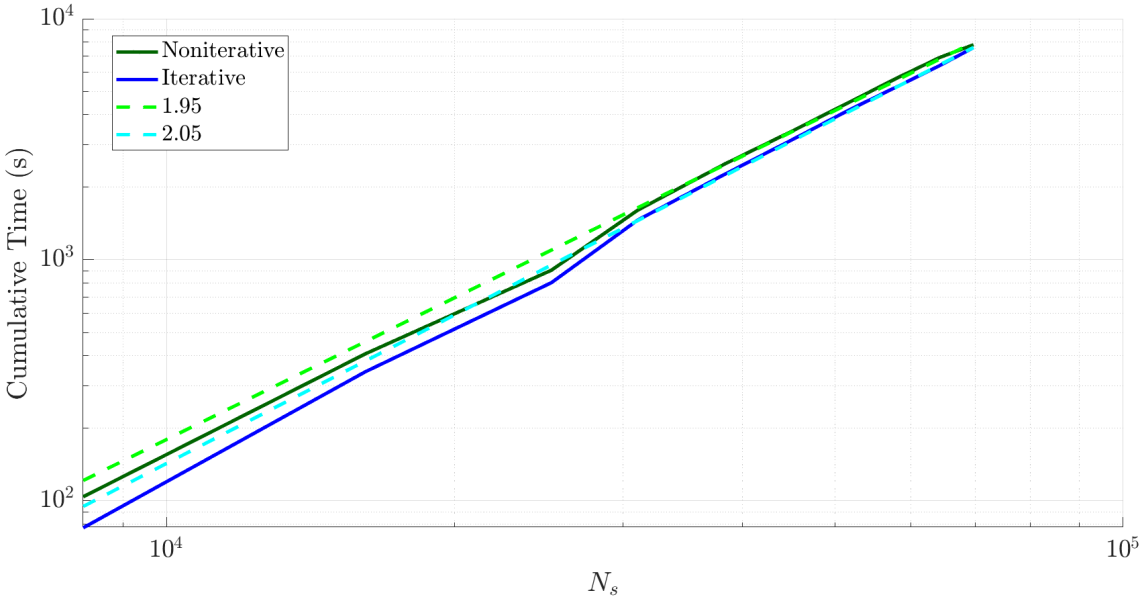


Figure 90: Cumulative time taken by the preprocessor in choosing 1000 base nodes over varying surface node numbers

Base Node to Base Node Influence Matrix $\mathbf{\Phi}_b$

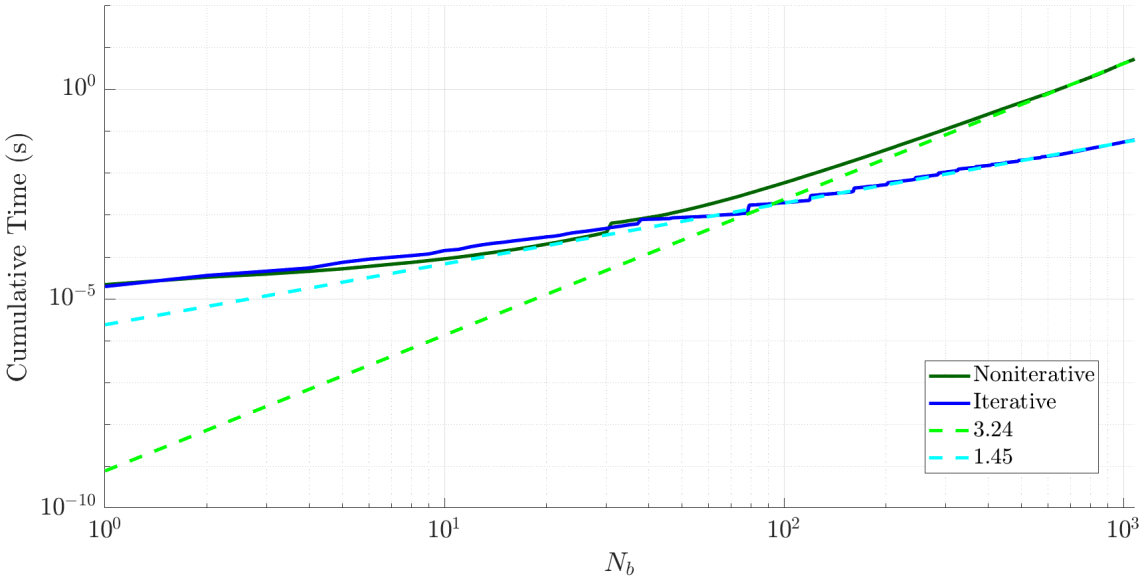


Figure 91: Cumulative time calculating $\mathbf{\Phi_b}$ in choosing N_b base nodes

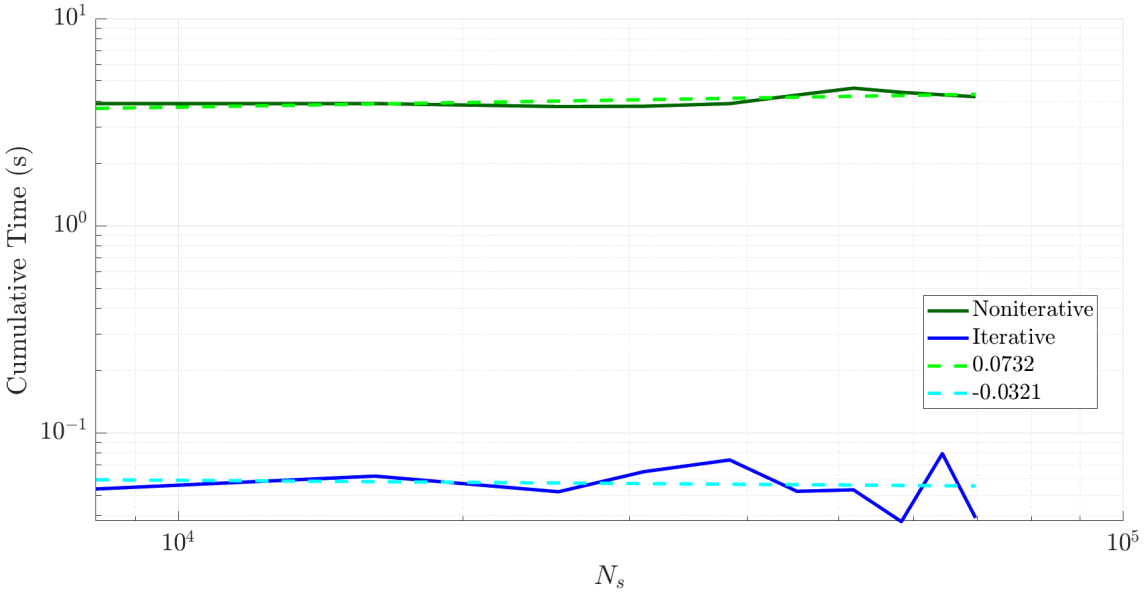


Figure 92: Cumulative time spent calculating $\boldsymbol{\Phi_b}$ in choosing 1000 base nodes over varying surface numbers

Base Node to Base Node Influence Matrix $\boldsymbol{\Phi}_b^{-1}$

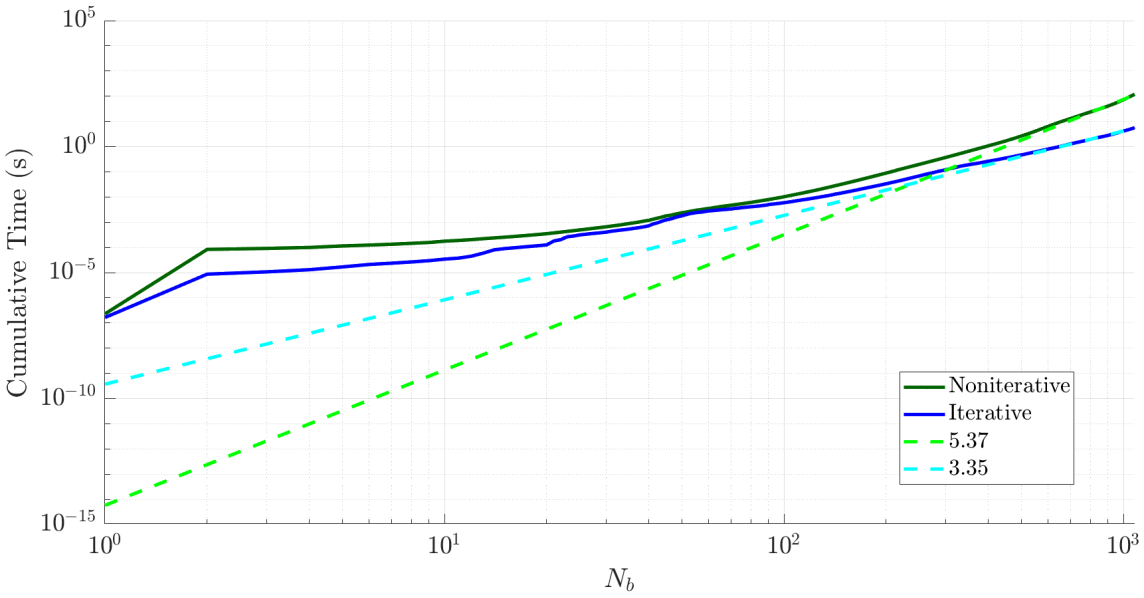


Figure 93: Cumulative time calculating $\boldsymbol{\Phi_b^{-1}}$ in choosing N_b base nodes

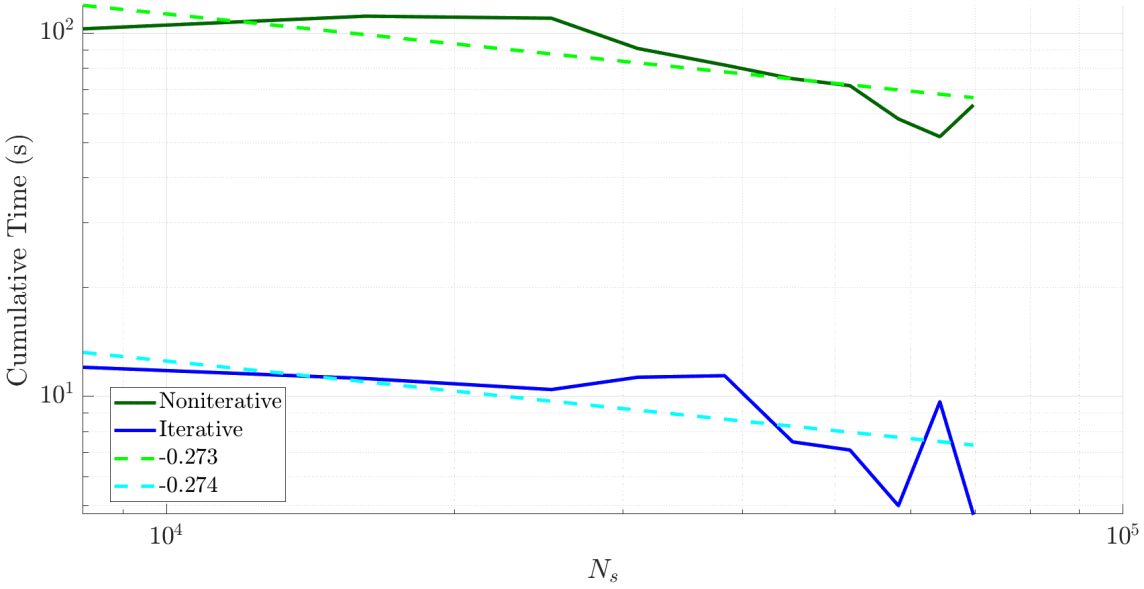


Figure 94: Cumulative time spent calculating $\mathbf{\Phi_b^{-1}}$ in choosing 1000 base nodes over varying surface numbers

Base Node to Refinement Node Matrix $\mathbf{\Phi}_r$

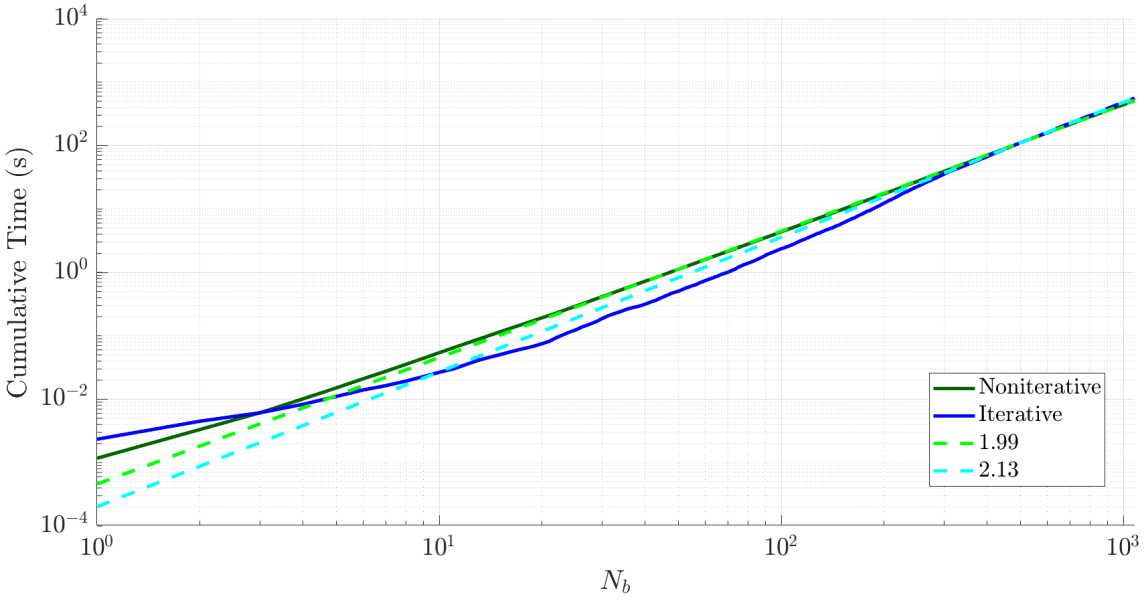


Figure 95: Cumulative time calculating $\mathbf{\Phi_r}$ in choosing N_b base nodes

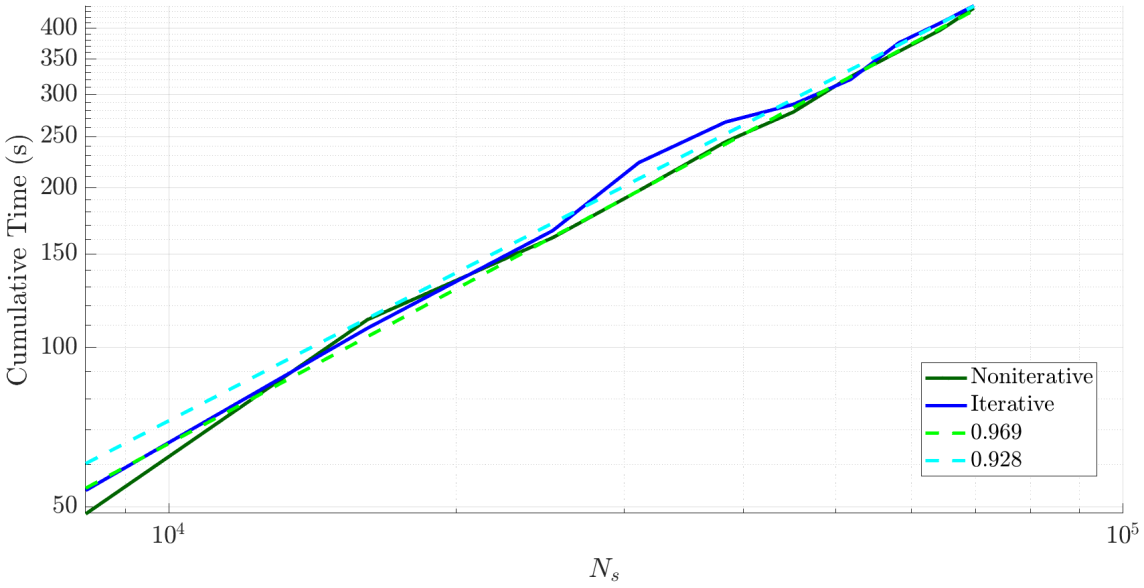


Figure 96: Cumulative time spent calculating $\mathbf{\Phi_r}$ in choosing 1000 base nodes over varying surface numbers

Refinement Node to Refinement Node Matrix L

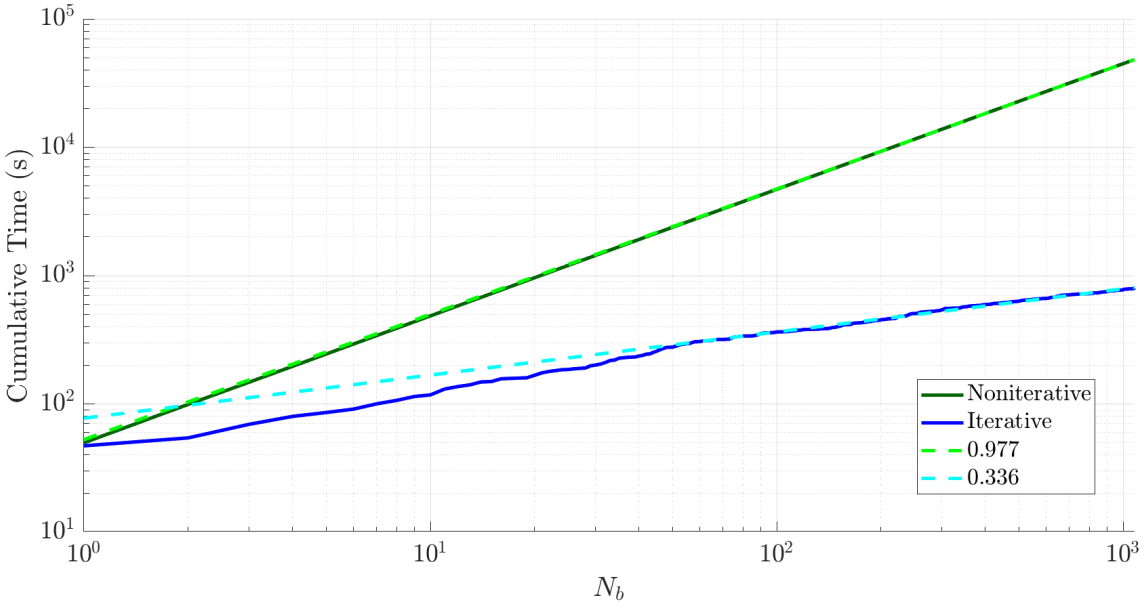


Figure 97: Cumulative time calculating $\mathbf{L}$ in choosing N_b base nodes

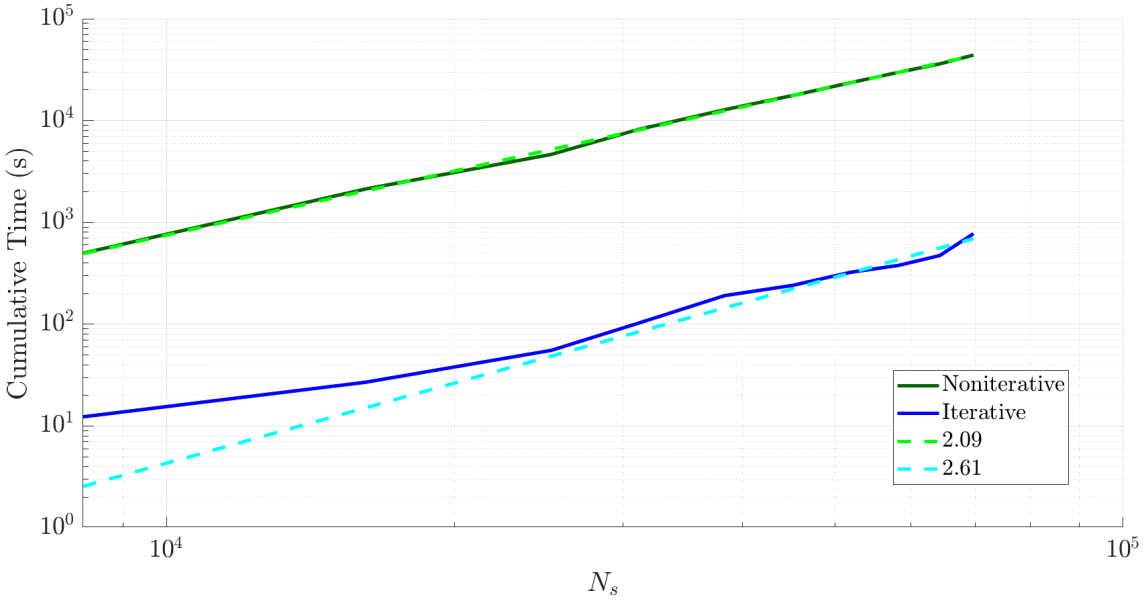


Figure 98: Cumulative time spent calculating **L** in choosing 1000 base nodes over varying surface numbers

Base Node to Volume Node Matrix $\mathbf{\Psi}_b$

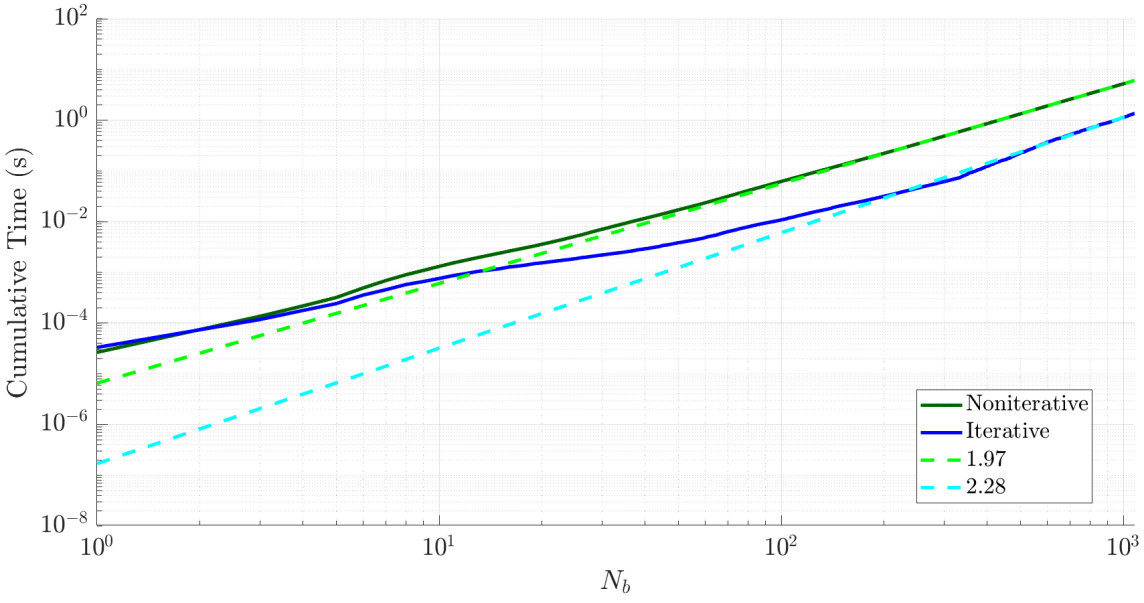


Figure 99: Cumulative time calculating $\mathbf{\Psi_b}$ in choosing N_b base nodes

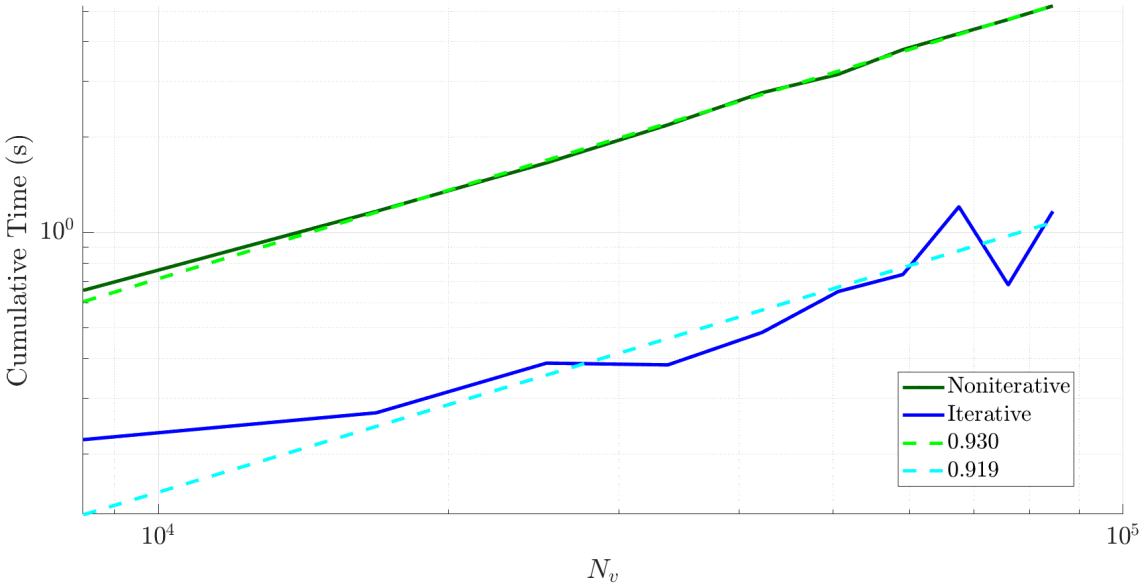


Figure 100: Cumulative time spent calculating $\boldsymbol{\Psi_b}$ in choosing 1000 base nodes over varying surface numbers

Refinement Node to Volume Node Matrix $\boldsymbol{\Psi}_r$

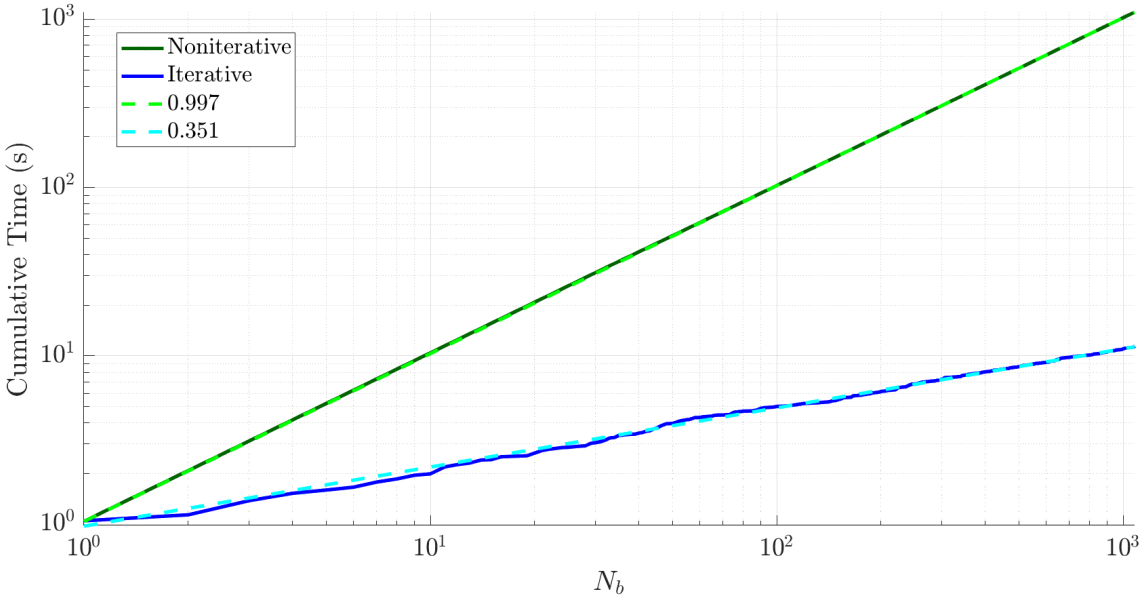


Figure 101: Cumulative time calculating $\boldsymbol{\Psi_r}$ in choosing N_b base nodes

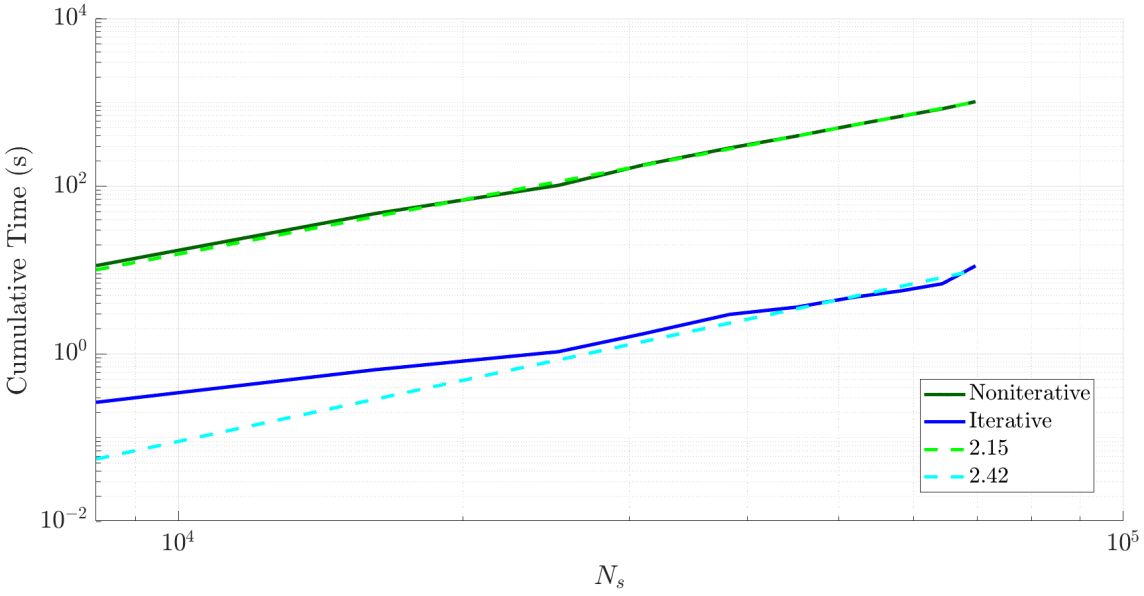


Figure 102: Cumulative time spent calculating $\boldsymbol{\Psi_r}$ in choosing 1000 base nodes over varying surface numbers

Total Costs

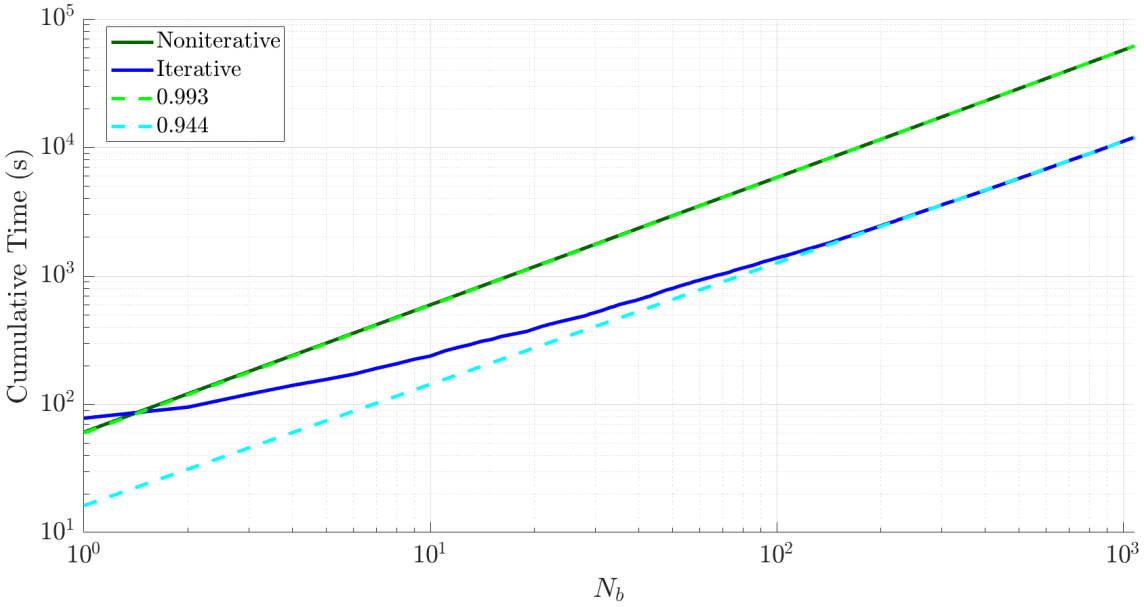


Figure 103: Cumulative time choosing N_b base nodes

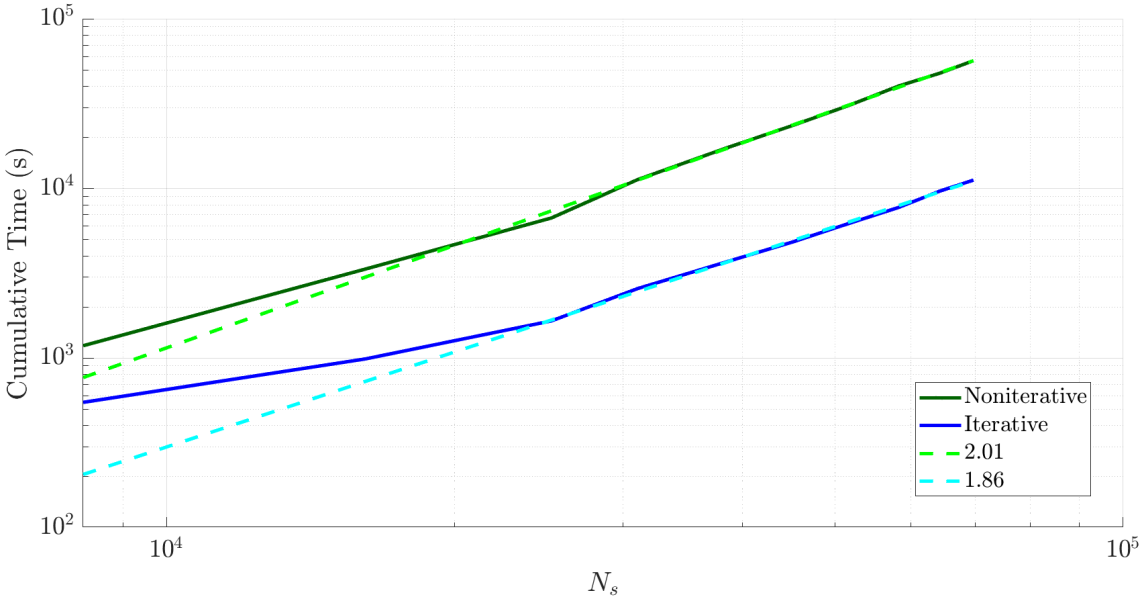


Figure 104: Cumulative time spent choosing 1000 base nodes over varying surface numbers